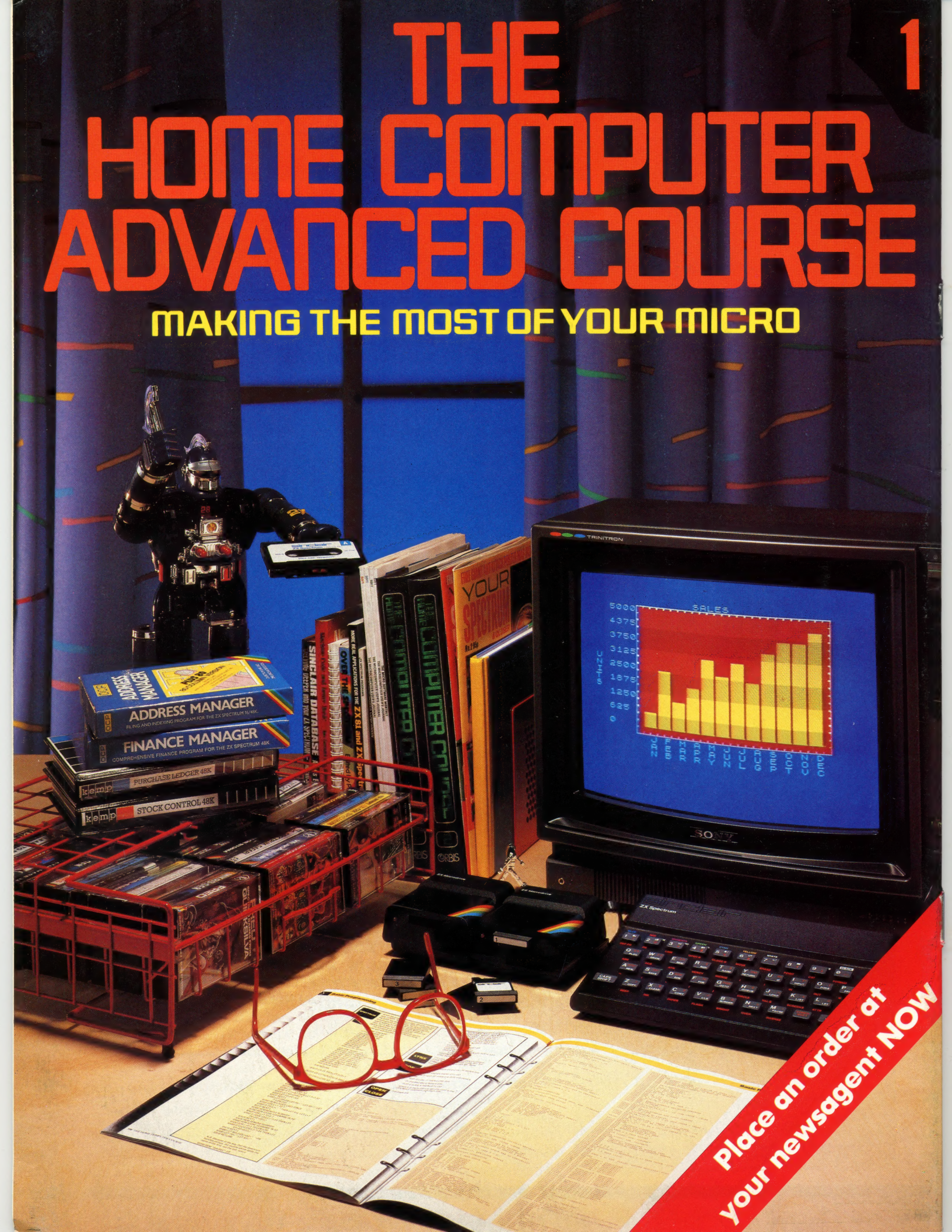
THE HOME COMPUTER ADVANCED COURSE

1

MAKING THE MOST OF YOUR MICRO



Place an order at your newsagent NOW

CONTENTS

APPLICATION

FIT FOR THE FUTURE An overview of the increasing relevance of computers to our everyday lives — 1

HARDWARE

BEST OPTION An introduction to disk drives, and how they work — 4

COMMODORE 64 We examine this popular home computer and look at its portable counterpart, the SX-64 — 10

SOFTWARE

ATTACKED BY ANTS A maze-chase game universally acclaimed — 6

COMPUTER SCIENCE

THE ALGEBRA OF DECISION MAKING The first step towards understanding program design — 8

JARGON

FROM ACCESS TO ADA The first part of a glossary of computing's terms — 13

PROGRAMMING PROJECTS

THE SPECTRUM OF ZX BASIC A look at the characteristics of this BASIC dialect — 14

MACHINE CODE

INTRODUCING FIRST CONCEPTS Commencing a course of instruction in this lowest common denominator of computer programming — 16

PROFILE

BILL GATES – SETTING THE STANDARD An insight into one of the world's most prolific suppliers of software — 20

Next Week

- **We continue our BASIC conversion course for Spectrum users, concentrating on functions and control structures.**
- **Widely acclaimed as the best of the microcomputer-based wordprocessing packages, Micropro's Wordstar can turn any CP/M based machine into a very powerful text processor.**
- **Amongst the educational uses of computers, one application is particularly attractive to the home user – examination revision software. We look at this fast-expanding field.**

Your special offer binder order form will be with Issue 5.

Overseas readers: this special offer applies to readers in the U.K., Eire and Australia only.

COVER PHOTOGRAPHY BY IAN McKINNELL TRINITRON COURTESY OF SONY SUPERROBOT 28 COURTESY OF HARRODS

Editor Jonathan Hilton; **Consultant Editors** Gareth Jefferson, Richard Pawson; **Art Director** David Whelan; **Deputy Editor** Roger Ford; **Production Editor** Catherine Cardwell; **Staff Writer** Brian Morris; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Sub Editors** Robert Pickering, Keith Parish; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Researcher** Melanie Davis; **Contributors** Lisa Kelly, Steven Colwill, Martin Hayman; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© 1984 by Orbis Publishing Ltd: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Details of how to obtain your binders (and of our special offer) are in issue 5. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

PAUL CHAVE

FIT FOR THE FUTURE

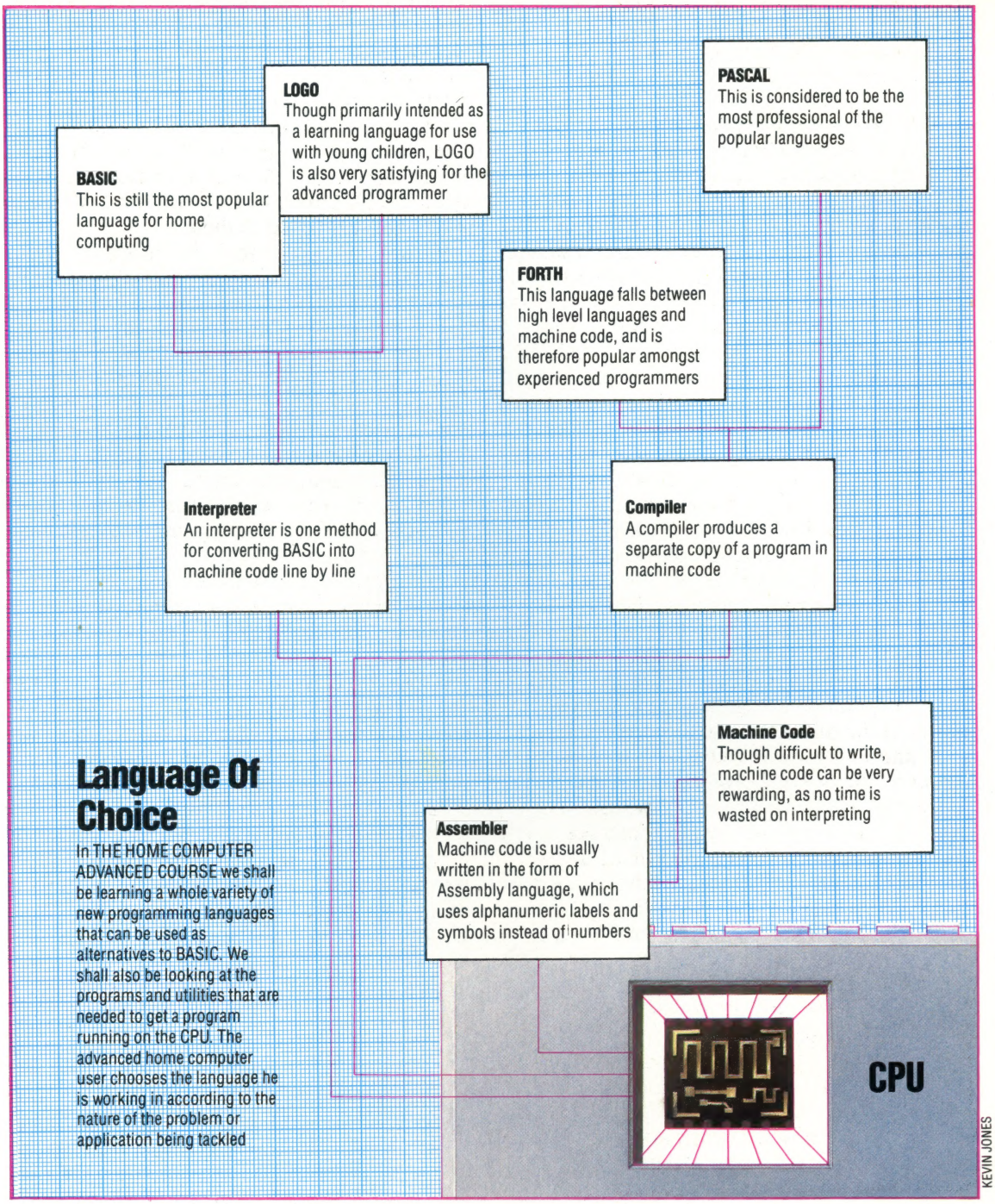
Most people over school age find learning a new subject from basic principles a daunting prospect, especially when the entire concept of that subject bears little relation to anything in their present experience. So how does the newcomer to the field of computing come to grips with the complexities of the subject?

It's difficult enough for the computer professionals – the operators, the programmers and systems analysts – to keep up with all the latest developments in computing, but at least these people can expect support from their employers and the suppliers of the equipment with which they work. Where do the amateurs and the hobbyists who must educate themselves in their own time and at their own cost, turn for unbiased advice?

What to do first? Go out and buy an inexpensive computer? There are so many, it's difficult to decide on a particular model without reliable advice. And having taken the plunge, what next? Perhaps the machine in question offers more than one programming language. Which one is most applicable to the user's needs? Which software packages offer the best value for money? Cartridges or programs on tape? Should the user buy a disk drive? Or will a cassette tape drive be sufficient? Do you really need a printer now, or should you wait until more refined printers come down in price? If you have children, how do you cope with their constant desire to play game after game, if you feel that the point of the investment was to foster computer literacy and help them with their school work?

The revolutionary advance of computer technology has been so fast and its encroachment into our everyday environment – home, office, factory and car – so comprehensive that many of us endow it with a mystique. It is, after all, not

Advance And Be Recognised
In the course of less than a decade, the computer has become an integral part of the fabric of our society — and every year changes and developments become more and more extensive. Keeping abreast of the situation demands a considerable effort, but for the newcomer the task is even more daunting. A well paced, comprehensive home study course is a great help


LOGO
Though primarily intended as a learning language for use with young children, LOGO is also very satisfying for the advanced programmer
PASCAL
This is considered to be the most professional of the popular languages
BASIC
This is still the most popular language for home computing
FORTH
This language falls between high level languages and machine code, and is therefore popular amongst experienced programmers
Interpreter
An interpreter is one method for converting BASIC into machine code line by line
Compiler
A compiler produces a separate copy of a program in machine code
Machine Code
Though difficult to write, machine code can be very rewarding, as no time is wasted on interpreting
Assembler
Machine code is usually written in the form of Assembly language, which uses alphanumeric labels and symbols instead of numbers
CPU
KEVIN JONES

Language Of Choice

In THE HOME COMPUTER ADVANCED COURSE we shall be learning a whole variety of new programming languages that can be used as alternatives to BASIC. We shall also be looking at the programs and utilities that are needed to get a program running on the CPU. The advanced home computer user chooses the language he is working in according to the nature of the problem or application being tackled

easy to come to terms with a 'motive force' that is invisible in action, unlike the satisfying spectacle of, say, a car engine or a hydraulic pump in operation.

True, more and more people are being trained to operate computers and computer terminals. But there is a huge difference between training and education. Training implies learning a task by rote. Education allows a leap beyond the bounds of the mere task at hand into a broad understanding of how the systems work, their potential and limitations.

To many people working in the computer industry and in schools and colleges the answer seems to be a planned course of computer education presented in such a way as to be understandable to all from the outset. Individual instruction manuals for specific machines cannot provide a balanced overview that relates one type of computer to another. Nor will they point out the pitfalls inherent in the multiplicity of machines available, or advise you fully on how to make the most of your purchase. After all, what sort of manufacturer is going to give free publicity to his rivals' products?

Following a properly planned home study course, perhaps backed up by a weekly session at an Adult Evening Institute (many of which offer introductory courses in computing and computer programming), is a convenient and inexpensive way to a sound education in computing.

The object of such a course should not simply

be to learn how to program and operate a home computer, but to gain a wider appreciation of how computers are used in everyday life. As well as providing instruction in programming and basic systems analysis, it should offer an overview of all the computers in use at the moment rather than concentrate on the machine one happens to be using. It should introduce the peripherals and extras available for all of them, with an explanation of their operating principles. To place the computer in context, one must examine in depth the tasks to which it is now applied and the software that makes those applications possible. Finally, the course should include elements of formal logic, number systems and something of the history of computing and computers. In short, a home study course should cover all the topics that would be dealt with in a conventional course in computer studies.

In THE HOME COMPUTER ADVANCED COURSE we have set out to provide the material for just such a course. Building on the average home computer user's knowledge of BASIC and some machine-specific experience of computer graphics and sound synthesis, we aim to take you through the other high-level languages found in microcomputers – PASCAL, FORTH, LOGO and C, for example – and to provide grounding in machine code programming, the key that unlocks the power of the microprocessor.

A knowledge of machine code enables us to examine the ways in which the higher level languages are defined. Then, when we have studied the way in which compilers and interpreters work, we can amalgamate these two branches of knowledge to start defining our own language and writing a compiler for it.

We won't neglect BASIC, however. We'll look at the refinements of the language and work through projects that will result in the generation of useful applications software and screen-based and Adventure games.

In addition to the internal functions of the computer, we'll explore file-handling methods, both on tape and on floppy disk, using the experience gained in defining data structures and hierarchies within the computer's internal memory. In this way we can expand the capacity of even the smallest home computer into a serious information processing system.

Bearing in mind that it's not enough to study a subject in isolation, we will consider in depth the wide choice of software packages now available – spreadsheets, word processors, database managers and the like – with a view both to understanding their operation and methods and to learning more about professional programming techniques, in order to include these in our own programming.

Some attention will be given to basic electronics, examining the function and design of individual components and the ways in which they are combined to make up computers and their peripherals. We'll look at the machines themselves, too: the popular microcomputers, both for home and business use, and their peripherals, examining their price and specification, and assessing their impact on computing in general. We won't neglect the human side of the computer industry, however. The people who design the software and build the machines, and even the computer users who have made a contribution to the field, will have space in the course devoted to them.

If you are interested in learning about computers with a view to increasing your employment opportunities, then a home study course can be an effective replacement for the first module or two of a formal course in computer studies. Because it allows the student to proceed at his or her own pace, it is of equal value to the fast learner, as well as those who perhaps need a little more time to come to grips with what is, after all, a complex subject.

Finally, if you simply wish to be better informed about a technology that is set to change society in the course of your lifetime, then THE HOME COMPUTER ADVANCED COURSE offers a comprehensive guide. In addition to the fundamentals of computer study, we shall be examining the impact of the new technology on society at large. How will the advent of computers in our everyday lives change the way people relate to each other? What political changes will result from an 'information explosion' made possible by the low-cost microprocessor? It is difficult to obtain reasonable answers to these questions. Newspaper articles and television programmes tend to trivialise them, many computer publications seem to make them more complicated than they need be. THE HOME COMPUTER ADVANCED COURSE sets out to give you access to the essential information to answer them for yourself.

A Leap Forward
Announced to the world's press at the beginning of 1984, but not scheduled for delivery until well into the spring, Sinclair's Quantum Leap broke that company's long association with the Z80 microprocessor. Fitted instead with a version of Motorola's 32 bit 68000, it has 128 Kbytes of RAM (with a further 512 Kbytes available), and two QL Microdrives built-in. Also abandoned is Sinclair's idiosyncratic single-key-entry BASIC

IAN McKINNELL

BEST OPTION

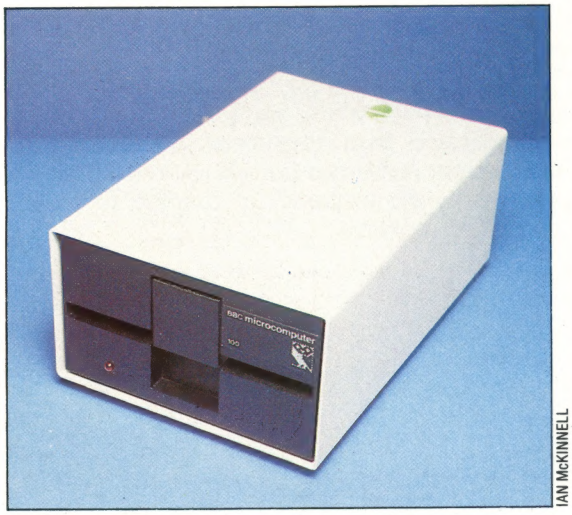
Until recently, floppy disk drives and so-called stringy floppies were beyond the budget of most home users, but advances in disk technology have reduced the relative cost of purchase, while the advent of the Sinclair Microdrive has bridged the gap. In view of the power of such devices it is worth looking at them in some detail.

DANGEROUS CONTACT

Remember not to put floppy disks close to anything that contains a magnet. Even something as seemingly innocuous as the telephone contains electromagnets (they are used to ring the bell), and a domestic hi-fi speaker has very powerful ones indeed

Microcomputers are highly versatile tools for manipulating data. However, data manipulation is of little use without a means of storing information when a particular set of data is not required for the moment or when the computer is switched off. This can be achieved in a number of ways. Anyone aware of the real potential of home computing will have acknowledged the limitations of the ROM cartridge and ordinary cassette tape as methods of permanent storage and will wish to investigate the more sophisticated facilities of magnetic disks.

But before discussing the merits of disks we will consider the alternative systems.



IAN McKINNELL

BBC Disk Drive
Before disk drives of this type can be used with the BBC Model B, the DOS (Disk Operating System) ROM must be installed in the machine itself. 'Intelligent' disk drives, on the other hand, come equipped with a DOS chip already on-board

CARTRIDGE

This method of storage is of little use to the programmer. Most cartridges contain a type of PROM (Programmable Read Only Memory) that provides only a means of inputting data to the computer, usually in the form of games written in complex and lengthy machine code, or extra facilities such as extensions to BASIC. It is possible, however, for cartridges to contain Electrically Erasable PROMs (EEPROMs) that can be written to and read from in a similar manner to internal RAM but which are 'non-volatile' in that the information is retained when they are removed from the computer or the computer is switched off. Similarly, cartridges are available for some computers containing low-power CMOS (Complementary Metal Oxide Semiconductor) RAM chips that retain stored information via a battery contained within the cartridge.

The main argument against EEPROM and CMOS RAM storage is that they are expensive — collecting a modest library of such cartridges would cost at least as much as an appropriate floppy disk drive.

CASSETTE TAPE

Originally provided because disk drives were very expensive, cassette tapes are still by far the most popular storage media, mainly because they are cheap, freely available and portable audio cassette players and tape cassettes are familiar to most people. Usually any cassette player of reasonable quality will suffice, although some manufacturers — notably Commodore and Atari — only allow you to use their own specially designed units.

Programs and data are stored in binary form as sequential files via the cassette unit's normal record facility, using different tones to represent 0s and 1s. Normally, identified information such as the file name (and possibly the internal memory address from which the file is copied) is recorded first, followed by the file itself, one bit at a time in one-byte blocks that are further formatted into 256-byte segments. Many computers incorporate an error-checking facility in each segment known as a 'checksum', which can be compared with calculations made within the computer during verification to ensure that there have been no recording errors.

Typical commands are SAVE to record files and LOAD to play back and retrieve them. Some systems provide additional cassette commands for various special functions, including a facility to read a tape and produce a catalogue of the file names stored, and command formats for storing and retrieving different types of data.

The low cost and easily understood command format of tape cassette storage is offset by a number of major inconveniences:

1. In the majority of cases the user is required to operate the cassette unit controls manually for storage and retrieval and this often demands careful timing of button pressing and accurate volume setting.

2. As information is stored sequentially, retrieval of a specific file (except in the case of the software-controlled Hobbit cassette recorder and the Epson HX-20's built-in micro cassette) involves either careful monitoring of an accurate tape counter (if one is supplied!) to enable fast

forward/rewind to a point just before the desired file, or a search by the computer for the file name from the beginning of the tape. Sequential storage also means that it is impossible to store data efficiently that needs to be read in small sections from any point in a file without processing the whole file. The type of storage that can achieve this is known as 'random access' and is necessary for any effective database filing system such as address listings or stock control entries.

3. The above, in conjunction with the small number of bits that are stored/retrieved per second using cassette storage — typically between 300 and 1,200 bits — means that a cassette tape system is excruciatingly slow in operation. Quite small programs of, say, five Kbytes could take between one and three minutes to load or save. This also means that it is inconvenient to make back-up copies of programs, although this is highly recommended.

4. Even when it has been recorded correctly in the first instance, data can be corrupted after an unpredictable number of replays, owing to wear by the tape head.

5. Because the characteristics of cassette players can vary from manufacturer to manufacturer, data recorded on one model may not play back on another. In addition, cassette tape is frequently damaged by the crude tape transport systems of many portable cassette units and breaks easily.

FLOPPY DISK

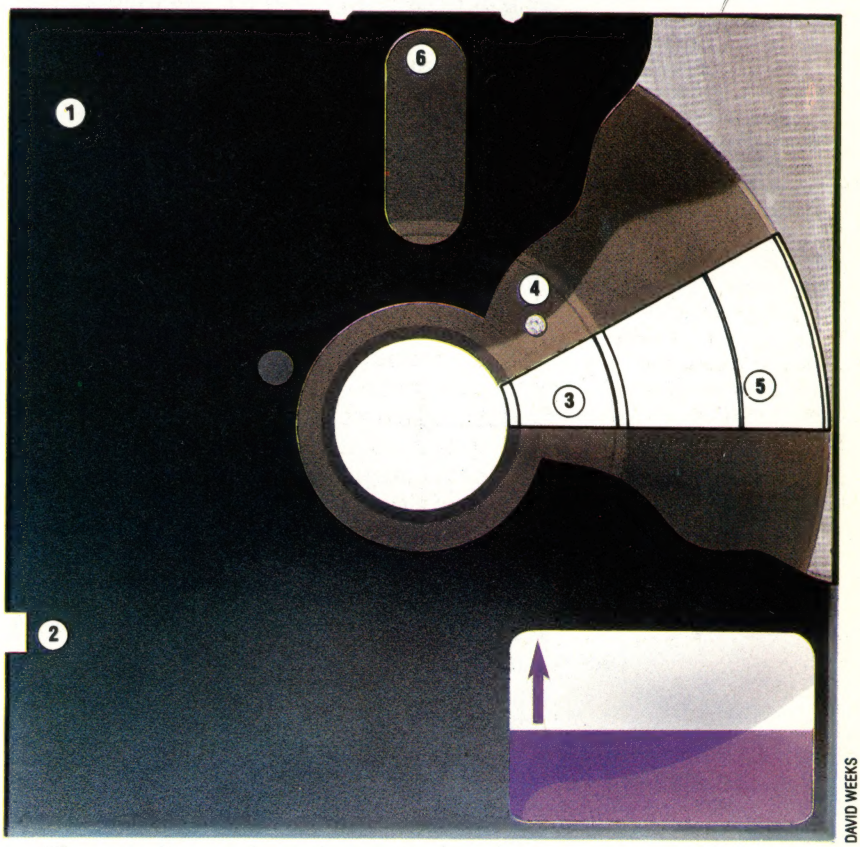
Compared with the cassette and cartridge storage systems, disk storage has few major drawbacks. Floppy disk drives are complex and delicate in their construction, and expensive — from £150 upwards. Floppy disks themselves are also costly at between £2.50 and £4 each. But the user gains a reliable, flexible and fast means of storing large amounts of data, operating at 50 to 200 times the speed of tape storage and retrieval.

All disk drives have a form of Disk Operating System (DOS), which contains a routine that formats the distribution of information on a disk into tracks. There are usually between 35 and 80 tracks per side, each track divided into a varying number of arcs called sectors. There are fewer sectors on the shorter tracks near the middle of the disk than on the long outer tracks. Each sector consists of a block of data, usually 256 bytes.

The DOS 'remembers' where all the information contained on the disk is stored. This is usually achieved by the creation of a Block Availability Map (BAM), either stored on the disk or held in memory, and a catalogue or directory. The BAM holds a record of the blocks currently in use and those free for new storage. The catalogue is a list of the file names, file types and track and sector locations for each file. It is usually held on the central track and can be loaded into computer memory for reference. The DOS positions the read/write head after reference to the BAM, and catalogues and manages the storage and retrieval of data.

The layout of the information in tracks and sectors and the accurate positioning of the read/write head enables the DOS to offer random access filing. Data can be recorded and extracted in chunks as small as a byte at a time, if required. In broad terms, differences between disk drives are confined to the amount of data that can be stored — typically between 100 and 400 Kbytes; the speed at which data can be transferred; and the means by which the user can control storage and retrieval using DOS.

In A Spin
Floppy diskettes are composed of Mylar, or a similar stretch and tear resistant plastic sheet, coated with a metallic oxide capable of holding a magnetic charge. Enclosed inside a protective square plastic envelope, the disk is spun from the hub. The recording surface is accessible to the read/write head through the slot shown at the bottom of the illustration



DAVID WEEKS

1 PROTECTIVE ENVELOPE
2 PROTECT/PERMIT SLOT
3 SECTOR
4 REGISTRATION HOLE
5 TRACK
6 ACCESS SLOT

There are three main methods of implementing a DOS. The most efficient is to include it in ROM form within the disk drive, under the control of the drive's own microprocessor with associated RAM. This is known as an 'intelligent' disk drive; on receipt of an instruction from the central processor it can process complex disk-handling routines independently, allowing the processor to continue running a program. All current Commodore disk drives are intelligent in this manner and use no internal computer memory in operation.

A more popular system is the type that loads the DOS from disk into computer RAM on command or automatically when the computer is switched on. The third method includes a form of DOS in the computer's own operating system. Spectrums have this facility and Acorn Computers supply a DOS for the BBC Micro called the Disk Filing System that provides limited disk control. Disk-handling routines include SAVE and LOAD commands, a CAT (or directory) command, a command to format a disk (or tape cartridge) and various random access and sequential file creating, handling and deleting commands.

ATTACKED BY ANTS

The significance of Quicksilva's Ant Atttack, a three-dimensional maze game designed for the ZX Spectrum with 48 Kbytes of RAM, lies not in its obvious graphic quality, but in the subtle application of the algorithm that generates the fabric of its maze-like playing ground.

Software writers and publishers have never been satisfied with the protection accorded them by the copyright laws – hence the many and various attempts to safeguard programs from being copied. The author of this game, Sandy White, has attempted to prevent his work from being plagiarised, by using another method – applying for letters patent on the software technique that produces the screen graphics. Since the 1977 Patents Act specifically denies protection of this sort to computer programs (noting that they cannot be considered to be inventions), one is led to the conclusion that the patent in question covers a mathematical formula or algorithm.

This in itself is interesting because one would not normally require a complex algorithm for a game of this sort. What is it about Ant Attack that requires a radically new approach to software protection?

1

1 My Hero!
On the first pass through the game, the 'victim' is conveniently placed adjacent to the gateway to the city. A quick hop over the protective wall, and the protagonist — male or female — is greeted with a cry of 'My hero — take me away from all this!'

2 Formi-dable Ant-iclimax
Sometimes, the fact that ants can't climb stairs is very useful indeed — though why our hero has climbed quite so high, one can only speculate. Climbing obstacles like this allows the protagonist to lob grenades at the attacking ants without fear of retribution, but remember that you are playing against the clock

Ant Attack is also unusual in that it is not descended directly from any arcade game. Most popular games for home computers have their roots in the conceptions of Atari, Taito and the other manufacturers of dedicated games machines. Ant Attack was conceived by a graduate from the Edinburgh College of Art who protests his ignorance of the arcade games tradition. Sandy White had never previously written games software and his efforts at market research were restricted to inquiring of friends what it was they liked about such games.

His remarkably forward-looking package was, surprisingly, rejected by Sinclair Research, who could not evaluate the videotape of Ant Attack that White sent them because, they said, they had no video cassette recorder!

The first novel feature of Ant Attack that a user will encounter is that it allows the player to choose the sex of the chief protagonist. And the first oversight follows hard on its heels. Whether you opt to be a girl or a boy, the opening frame of the game, which sets the scene in 30 or so words, explains how you hear a call of distress 'irresistable (sic) to a hero like you'. One can forgive the spelling mistake, but the program's inability to substitute 'heroine' for 'hero' is evidence of a lack of attention to detail. Further evidence is to come.

The protagonist, chased by monster ants, can defend himself (or, of course, herself) by throwing grenades. Unfortunately, there is no consistency in

2

COURTESY OF SOFT

IAN McKINNELL

the effect these grenades have on the ants. While this might result from a deliberate randomising factor, it is more likely to be the result of indiscriminate programming. Moving the protagonist anti-clockwise through 90 degrees is achieved by pressing the Spectrum's M key, and the Symbol Shift key next to it turns the figure the other way. The Spectrum's moulded rubber membrane keys do not give proper control over this transformation, which invariably results in frustration for the player.

It would appear that Ant Attack was developed in advance of the launch of Sinclair's Interface 2, which accepts two Atari-standard joysticks. The game would benefit greatly from being updated to utilise these peripherals, though it would need two joysticks to handle the command structure.

In addition to revolving the token, moving it forward, making it jump or throw grenades (you can also choose between four distances of throw), the player can choose one of four points of view —

each centred on the token.

It is this section of the program's graphics generation that sets it apart from most other games occupying less than 48 Kbytes. The transformation is virtually instantaneous, completely overshadowing the normal run of 3D graphics generators available for Spectrum. The ability to change points of view is essential to the game. Without it a considerable portion of the playing ground would often be hidden from view.

The author is understandably unwilling to reveal too much about the working methods that he and his collaborator Angela Sutherland have adopted. He does imply, however, that the playing ground is not, as one would expect, held as a 128 × 128 × 6 array. Evidence of this is apparent if, rather than entering the city, the player token is made to turn round and head off into the desert. After a short walk, he or she comes to another city, and then another, and so on.

And so to the object of the game itself. It is set in the City of Antescher (named by the game's authors in tribute to the Dutch artist and designer M. C. Escher, who drew ingenious delusive structures that were impossible to actually build). Standing outside its gates, you hear the cries of a person in distress. You jump over the low wall into the city and go off in search of the victim, jumping onto obstacles or turning to avoid them as you go. The city appears in isometric projection and no attempt is made to keep faith with perspective.

Only a small portion of the city is in view at any one time, the frame scrolling across as the figure moves left, right, up or down. The scrolling action is excellent, as is the animation of the figures. Full marks, too, for a good sense of humour in the treatment of the animation.

It soon becomes apparent that the city is populated by huge ants whose bite, though not immediately fatal, will cause death if you suffer enough of them. If an ant becomes aware of you, it will follow you. You can shake it off if you are skilled enough, otherwise you have to resort to the rather unreliable grenade. Don't throw it at the wall immediately in front of you, because you could blow yourself up.

On the first pass through the game the figure to be rescued is in full view opposite the gate. On successive passes it gets harder to find, and harder to reach. It is invariably located above ground level. The rescuer may jump up only one level at a time, so if the victim is not directly accessible from the ground – by a stairway, for instance – the rescuer is in real trouble. The only way is to wait till the ants attack at a suitable spot, paralyse one, and jump onto its back, using it as the first step up.

The rescuer can also get a 'leg up' in this way from the victim, should it be necessary – the ants won't attack the victim. The pass finishes when rescuer and victim are both outside the city.

Despite its few failings, Ant Attack is worthy of the accolades that greeted it when it appeared on the market just before Christmas 1983. It is a fine example to all would-be software authors.

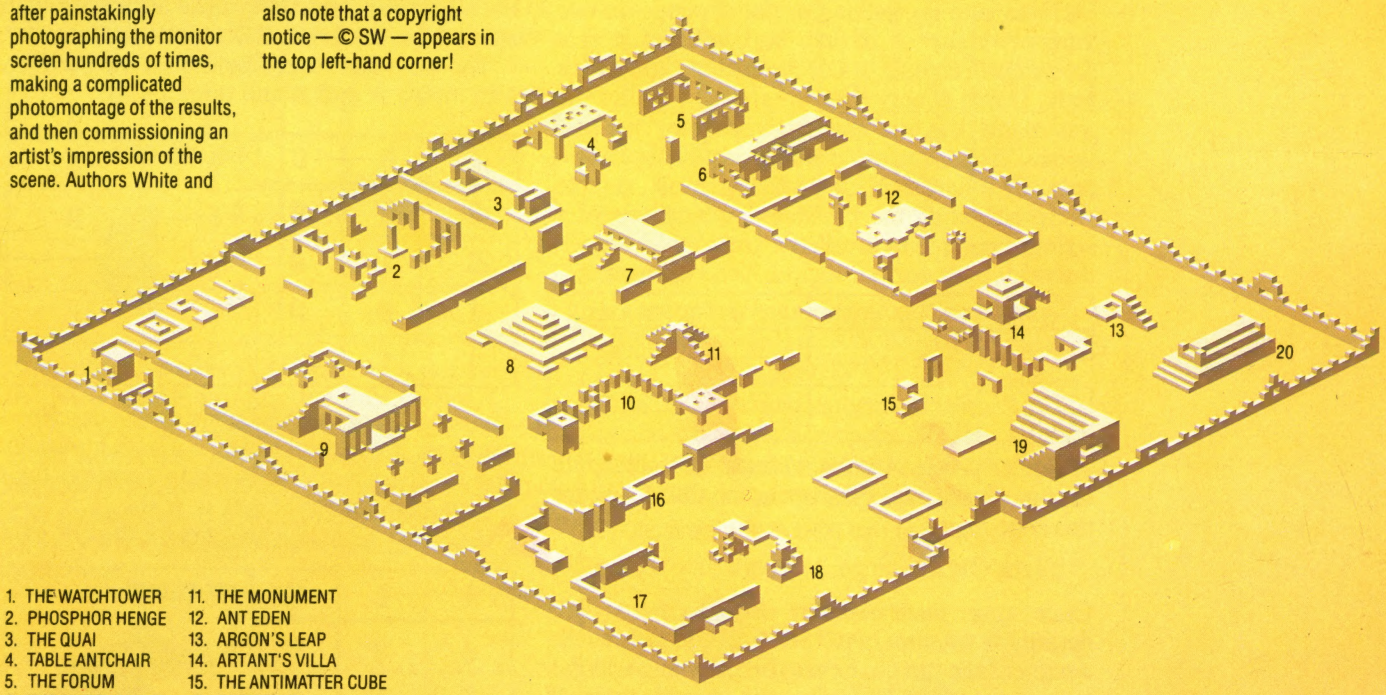
IAN McKINNELL

Master Minds

Ant Attack was a first attempt at commercial software writing for its author, Sandy White. Sandy, just 23 years old when the package first appeared in late 1983, had graduated from Edinburgh College of Art with a degree in sculpture when he conceived the notion of creating a games program for home microcomputers. A friend, Angela Sutherland, collaborated in the design of the structures that make up the city of Antescher

Riddle Of The Sands

This plan of the entire city of Antescher was constructed after painstakingly photographing the monitor screen hundreds of times, making a complicated photomontage of the results, and then commissioning an artist's impression of the scene. Authors White and Sutherland have given names to the chief structures, but also note that a copyright notice — © SW — appears in the top left-hand corner!



1. THE WATCHTOWER
2. PHOSPHOR HENGE
3. THE QUAI
4. TABLE ANTCHAIR
5. THE FORUM
6. THE ANTICHAMBER
7. SKAZ YANDOR
8. THE PYRAMID
9. THE ANCIENT
10. OXYMINE
11. THE MONUMENT
12. ANT EDEN
13. ARGON'S LEAP
14. ARTANT'S VILLA
15. THE ANTIMATTER CUBE
16. DROXTRAP
17. ADRIANT'S WALL
18. BONZAI WALK
19. THE SQUARENA
20. THE CRYPT

COURTESY OF YOUR SPECTRUM

KEVIN JONES

THE ALGEBRA OF DECISION MAKING

Computers carry out their given functions by passing a series of high or low voltages around electronic circuits. These voltages can be interpreted in terms of the binary digits (or bits) 1 and 0. Some functions, such as addition, require specially designed circuits to produce specific outputs for any given input. These are termed 'logic' circuits.

Boolean algebra, the branch of mathematics concerned with true/false logic, is the theoretical basis from which computer architecture is physically realised. The concepts and rules of Boolean algebra are few and easily understood.

In the first instalments of this course, we will study in detail the theoretical and practical aspects of logic circuit design, together with examples of the basic circuits at work inside your own home computer. The rules of Boolean algebra are based on three simple logical operations: AND, OR and NOT. These three logical operations conform closely to the way we use these words in everyday English. Look at this statement:

If it is fine AND it is a Saturday, David will go for a walk.

If David is to go walking or not depends on two things: whether it is fine, and whether it is a Saturday. In coming to a decision about going for a walk, David is only concerned with whether the statements 'it is fine' and 'it is a Saturday' are true or false. There are four possible combinations and only one will result in David taking a walk. A table which shows all the possible combinations of a series of statements is called a 'truth table'. Here is the truth table for our logical AND statement:

IT IS FINE	IT IS A SATURDAY	DAVID WILL GO FOR A WALK
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

A similar process can be undertaken to illustrate the function of the logical operation OR. Consider this statement:

If Jack OR Jill can go, John will go to the match.

Once again there are two conditions that will determine whether or not John goes to the match: whether Jack can go, *or* whether Jill can go. In the same way as the AND statement, we can construct a truth table for the OR statement. Since there are two conditions, each of which may be true or false, there are again four possible combinations. The truth table for the statement will look like this:

JACK CAN GO	JILL CAN GO	JOHN WILL GO TO THE MATCH
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

The third logical operation (NOT) performs a very simple function. Consider this statement:

If it is NOT dark then I will go out.

This time the only condition to consider is whether it is dark. This may be true or false; hence there are only two possible conditions for our truth table.

IT IS DARK	I WILL GO OUT
FALSE	TRUE
TRUE	FALSE

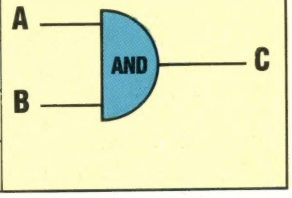
LOGIC GATES

The simple electronic devices that make up computer logic circuits are called 'logic gates'. The three simplest logic gates mimic the function of the logical operations AND, OR and NOT. These gates function by representing a TRUE condition by the binary digit 1, and the FALSE condition by the binary digit 0. So, for each logic gate we can construct a truth table showing all the input combinations together with the resulting output. Each gate has a circuit symbol associated with it and can be written as a Boolean expression.

The truth table and diagram for the AND gate with inputs A and B and output C is:

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

THE AND GATE

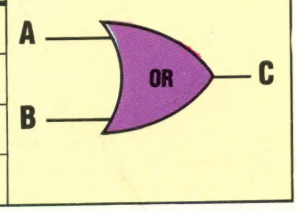


The function of the AND gate can be described in words as: 'the output will be 1 if both inputs are 1, and 0 otherwise'. The Boolean notation for the output from an AND gate is A.B.

The truth table and diagram for the OR gate is:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

THE OR GATE

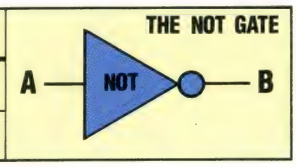


NOXIO 211

The OR gate can be described by the following statement: 'The output will be 1 if either or both of the inputs are 1'. The Boolean expression for the output from an OR gate is A+B.

Unlike AND and OR, the NOT gate has only one input and one output. The truth table is the simplest of the three:

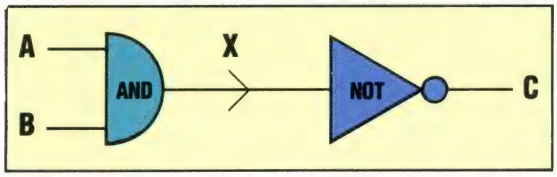
A	B
0	1
1	0



In words, the NOT gate is expressed as: 'the output will be the opposite of the input'. The Boolean expression for the output from a NOT gate is $\overline{A}$.

COMBINING LOGIC GATES

Just as several logical statements can be linked together, we can link together logic gates to make combinational and sequential logic circuits. These are in turn combined to produce the computer architecture. Any logic circuit can be represented by a truth table that describes what output can be expected for any possible combination of inputs. Look at this simple logic circuit:

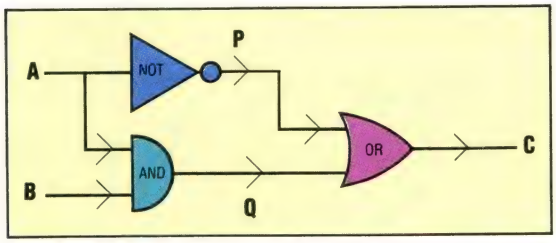


In this circuit there are two inputs, A and B, and one output, C. To help to construct the truth table for the circuit the output from the first gate has been labelled X. As there are two inputs to the circuit this means that there are four possible combinations of input.

A	B	X	C
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

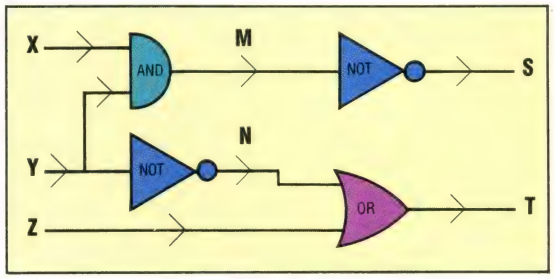
The output from the AND gate, X, is put through the NOT gate to produce the final output, C.

Here is a more complicated circuit and its truth table. Notice that, as there are only two inputs, the number of possible input combinations is still four. The second half of this truth table (columns P, Q and C) is a rearrangement of part of an OR gate truth table.



A	B	P	Q	C
0	0	1	0	1
0	1	1	0	1
1	0	0	0	0
1	1	0	1	1

The use of truth tables is not limited to two input and one output circuits but can be extended to any circuit. Here is an example of a three input, two output circuit.



As there are three inputs to this circuit we must consider eight possible combinations:

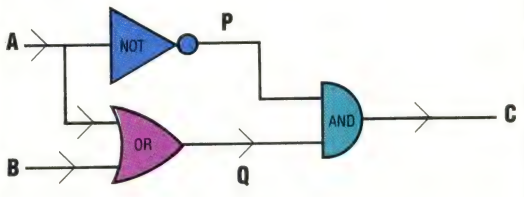
X	Y	Z	M	N	S	T
0	0	0	0	1	1	1
0	0	1	0	1	1	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	0	1

EXERCISE 1

1) Construct a truth table for the following situation: 'James may drive a car if he has passed his driving test OR he is accompanied by a qualified driver'.

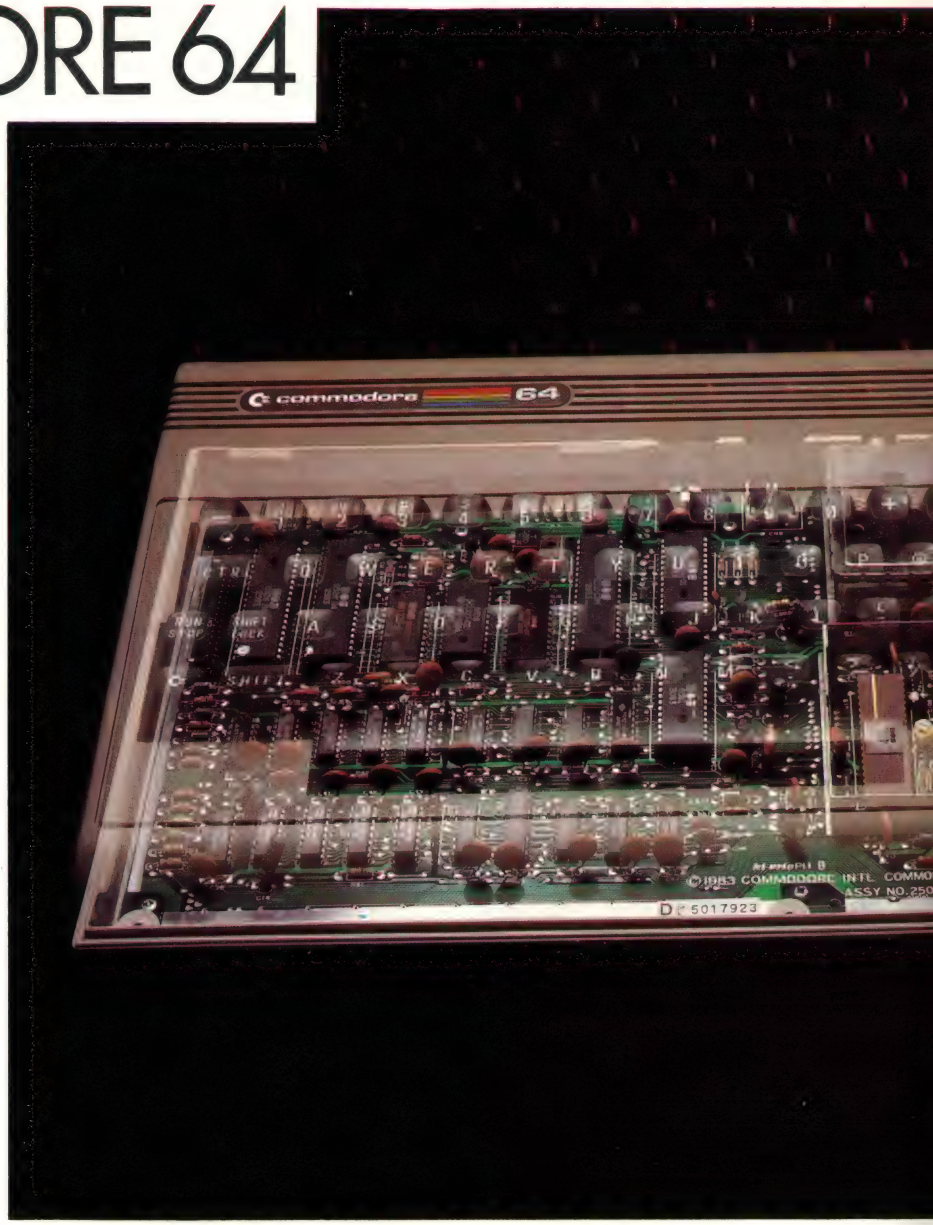
2) Construct a truth table for this situation: 'A program can be loaded into a computer if there is a cassette player OR a disk drive available AND the program is NOT written to run on a different computer'.

3) Construct a truth table for this logic circuit:



COMMODORE 64

At £200, the Commodore 64 contains a lot of hardware — 64 Kbytes of memory, sophisticated sound and graphics facilities. It is a very suitable machine for the serious home computer enthusiast, and with the addition of suitable peripherals, could be used for small business applications , too. The design makes use of 'bank switching' to squeeze the memory into the space available.



The physical similarity between the Commodore 64 and the Vic-20 is deceptive. Although there is a measure of software compatibility between the two, in hardware terms the 64 represents a considerable advance. Let's begin by looking at the 64 Kbyte of RAM from which the computer derives its name. This feature is a considerable advantage in selling terms since it was, until the advent of the 16-bit microprocessor, as much RAM as was available on any business microcomputer. However, there is a certain amount of difficulty associated with equipping a home computer with this much memory. Though an eight-bit microprocessor such as the widely-used 6502 can address a total of 64 Kbytes, this must include all the ROM and the input/output chips for controlling keyboard, screen and peripherals in addition to the RAM.

The answer lies in 'bank switching', a technique whereby sections of memory are switched into and out of the addressable memory map as they are needed. There is no theoretical limit to the total amount of memory that a computer can incorporate using this method, but because the microprocessor can still only address 64 Kbytes at

Box Of Tricks

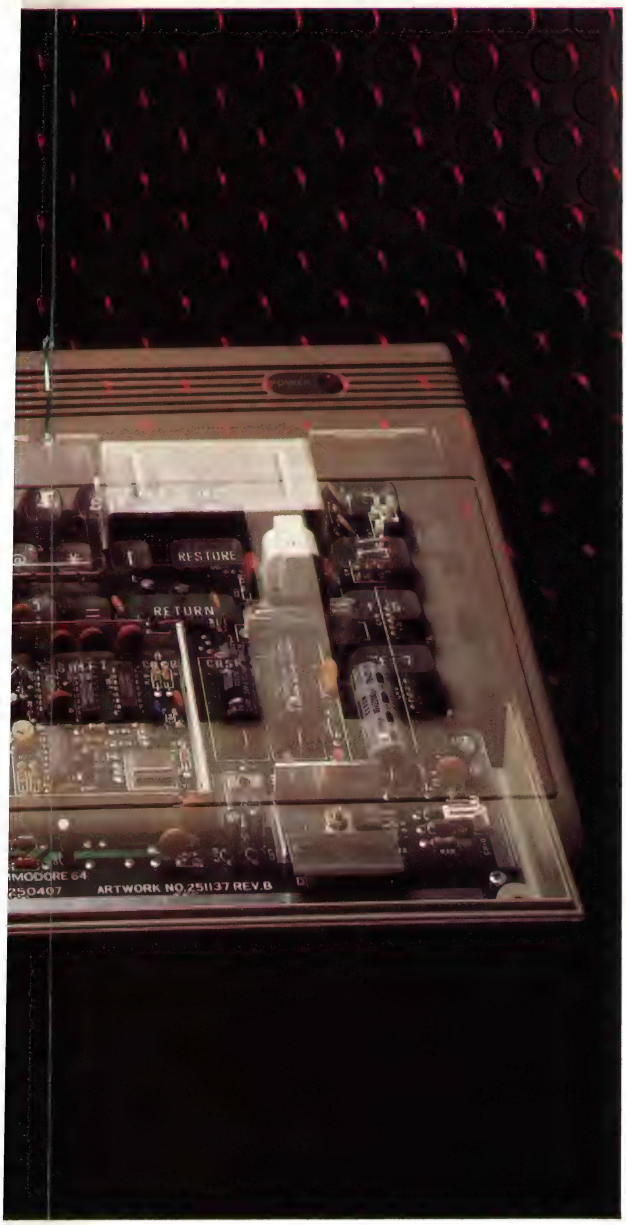
The SX-64 is a self-contained portable version of the Commodore 64, which can be purchased in a variety of different configurations. The most popular version features one disk drive (the space above can be used for storing diskettes) and a five inch colour monitor. The SX-64 will run disk or cartridge based software from the standard Commodore 64 without modification.

In many respects it is one of the best designed 'luggable' computers — a phrase coined to distinguish them from truly portable machines such as the Epson HX-20 and Tandy Model 100. The keyboard features fully sculptured keys with the graphics legends inscribed on the front, and it is detachable from the main unit. There is a slot in the top of the casing to take ROM cartridges; when not occupied the opening is covered by a flap to keep out dust.

The casing itself is both rugged and compact, resembling the portable test equipment used by service engineers, particularly in the way that the carrying handle doubles up as a stand. The handle is ridged to prevent it from slipping on the desk, though this makes it slightly uncomfortable to carry. Overall, the physical design is the best to have come from Commodore to date, and is marred only by the fact that the mains cable and plug cannot be stored anywhere inside the casing

SX-64 COURTESY OF COMMODORE

IAN McKINNELL



IAN McKINNELL

any one time, the more memory there is, the more switching of banks there will need to be, and that will subsequently reduce efficiency.

What this amounts to on the Commodore 64 is that if you want to run a program in BASIC, the ROMs containing the BASIC interpreter will need to be switched in, and this will reduce the amount of available RAM to 40 Kbytes (and system variables and screen RAM will still need to come out of this allocation).

Though bank switching has been added to quite a few home computers by way of a modification, it is achieved on the Commodore 64 by using a special microprocessor. The 6510 is very similar to the 6502 that has proved so popular in home computer design. The instruction set is identical, and it features an eight-bit data bus, 16-bit address bus, and various control signals. However, it also sports an eight-bit programmable input/output port. This means that there are eight additional pins on the chip, each of which can be set to 1 or 0, or can be used to read values placed onto them by an external device. Normally such ports are implemented by means of a special chip (called a PIO, PIA or VIA depending on the manufacturer), and a typical home computer will include several of these to handle the keyboard and peripheral ports.

The port appears as the lowest two memory locations in the map ($0000 and $0001). The former is for reading and writing the individual bits, while the latter location indicates whether each bit is set as an input or an output. Having this port built into the microprocessor means that the 6510 would be ideal for incorporation into numerous domestic devices — from dishwashers to programmable toys. On the Commodore 64, it is used to select between the banks of memory (see panel). You could do this with BASIC POKE statements, but there is a distinct possibility of 'crashing' the system, forcing you to reset the computer. Most memory switches are therefore

COMMODORE 64

PRICE
Approx £200

DIMENSIONS
404×216×75mm

CPU
6510

MEMORY
64K RAM, of which 39K is available for BASIC programs. 20K of ROM including the character generator

SCREEN
25 rows of 40 columns. In low resolution, 16 colours are available from the keyboard for characters, border and background. Maximum high resolution is 320×200 pixels. Up to eight sprites can be defined and used

INTERFACES
Joysticks (2) plus light pen, RS232 (adaptor needed), 8-bit parallel, cassette, serial (for disk and printer), composite monitor, audio input and output, TV, cartridges

LANGUAGES AVAILABLE
BASIC, FORTH, LOGO, 6502 Assembly language

KEYBOARD
Typewriter-style, with cursor keys and four programmable function keys

DOCUMENTATION
The computer comes with an adequate instruction manual, but to take full advantage of the functions, you should purchase the Programmer's Reference Guide, or one of the many independently published guides to the Commodore 64

STRENGTHS
Large standard memory. Sprite graphics. Sophisticated sound control. Quality keyboard. Good range of peripherals. More business software available than for most home computers

WEAKNESSES
Requires manufacturer's cassette unit. BASIC weak on useful commands (unless you purchase a cartridge add-on). Limited choice of graphics modes and resolutions. Disk unit slow

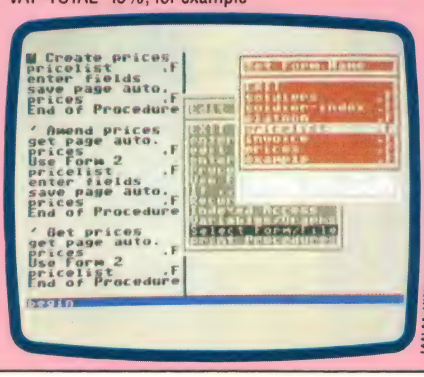
Business Mileage

A large proportion of the Commodore 64's software base can be said to have been inherited from its predecessors, the PET and Vic-20. The BASIC interpreter is more or less identical on all three machines, and there is much common ground in the disk operating systems, too. Because the business software developed for the PET range could only be used on Commodore's machines, it is hardly surprising that the software developers were so quick to take advantage of the potential new market opened up by the 64.

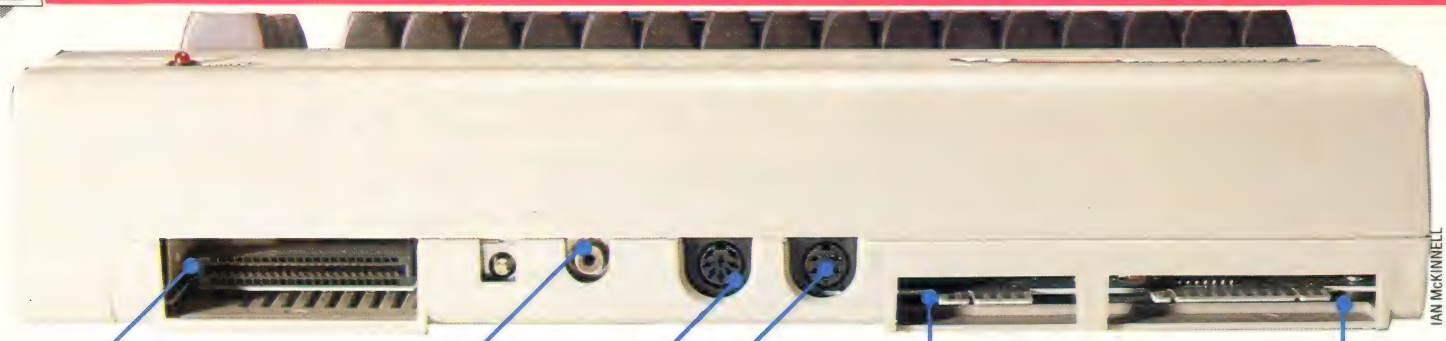
For business application, there is a wide choice of word processing packages, several of which have spelling checkers. Two of the most popular examples are EasyWrite/EasySpell from Commodore, and VizaWrite/VizaSpell. Two other popular packages, but without the spelling option, are Paperclip 64 and Wordcraft 40. The latter is different from most word processors, in that the screen displays the text in the format in which it will finally be printed, whilst most others display 'embedded controls' — single character symbols to signify a carriage return, or that a heading is to be centred on the page.

Spreadsheets are available starting from just a few pounds. One package, however, is worth special mention. At over £100 CalcResult is more expensive than most spreadsheets for low-cost micros, but it works in full colour, includes a facility for displaying barcharts of the figures in any column on the spreadsheet, and works three dimensionally. That is to say, several pages of memory can be held in memory at once; and it is possible to add together figures from all these sheets.

Magpie, too, is a fairly outstanding piece of software — falling into the category of applications generators. An application is defined by drawing the layouts for screen records and printed forms on the screen, and then specifying the relationships between the fields within those documents: VAT=TOTAL*15%, for example



IAN McKINNELL



IAN McKINNELL

Cartridge Port
If a ROM cartridge (up to 16 Kbytes) is plugged in here, it will effectively override any other memory that occupies the same locations. If the first nine bytes of the ROM contain a specified sequence of values then the program will 'automatically start' when switched on. This is how games cartridges work

TV Output
Unlike the Vic-20, the Commodore 64 contains a built-in RF modulator, so that the output can be connected directly to a TV

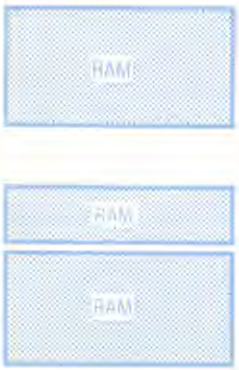
Audio/Video Socket
A composite video signal is provided to drive a colour monitor (though not an RGB monitor), and there is a separate audio output that can connect with a hi-fi system. There is also an audio input line that allows you to mix recorded music with synthesised sounds

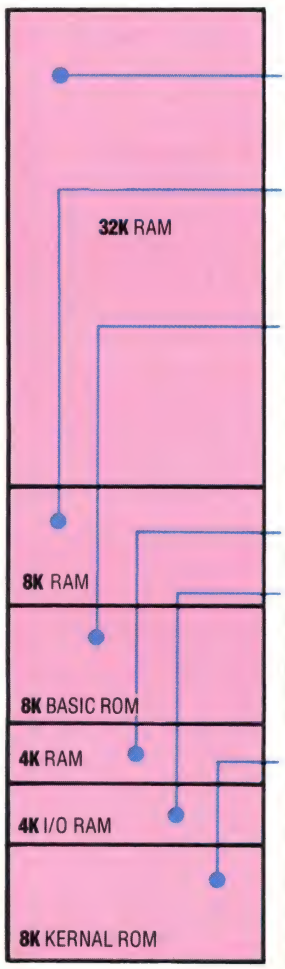
Serial Bus
This is a special interface designed by Commodore to drive several devices (including their disks and printers) simultaneously. The protocol is similar to the IEEE48 standard, except that there is just one (serial) data line instead of eight parallel ones

Cassette Port
All Commodore computers require the manufacturer's cassette unit. When it was first marketed, the Commodore system was faster and more reliable than a domestic unit. Now the opposite is true

User Port
This port has two functions. First, it can implement a full RS232 serial interface, though an add-on is needed to convert the 64's voltages to those used on most serial devices. It can also double up as a parallel port that can be used for experimentation

Memory Map
The 64K of available memory space is divided up into six zones, three of which are usually configured as RAM. The other three contain ROMs for the BASIC, the operating system, and the I/O chips, but for each one there is a 'shadow' area of RAM that can be switched in under software control. This is only practicable, however, when using machine code and the ROM isn't needed





Zone 1
Permanently contains 32K of RAM

Zone 2
Normally 8K RAM. If an 8K ROM cartridge is inserted this will override the RAM

Zone 3
Normally 8K ROM, containing the BASIC interpreter. It will be overridden if a 16K cartridge is inserted. Alternatively, 8K of RAM can be switched in to replace the ROM

Zone 4
Permanently 4K of RAM

Zone 5
Normally contains all the I/O chips and some RAM. Under software control, however, the ROM-based character generator can be switched in here

Zone 6
When the machine is switched on, this contains 8K of ROM, including the Kernal – a minimal operating system designed to be used on all future Commodore hardware. When using machine code it can be replaced with 8K of RAM

performed in machine code.

Three other chips between them account for the rest of the 64's features. There is a 6526 CIA (Complex Interface Adaptor), which is a more sophisticated version of the PIAs and VIAs previously mentioned. In addition to the usual programmable input/output lines, it includes timers and shift registers to convert between serial and parallel data. There is also a 24-hour clock with a programmable alarm, of which the BASIC interpreter appears to make no use at all.

The graphics and video display are handled by another chip, the 6566, which is a further development of the Video Interface Chip, from which the Commodore Vic-20 derives its name. This delivers different modes for both textual and high resolution graphics displays, and the sprite graphics have been well documented. Though it can handle only eight sprites at once (compared with 32 on the Memotech MTX512, for example), it is possible to simulate rather more. Sprites are defined as a block of bytes in memory, and their location is indicated by POKEing the address into the Vic-II chip's registers. It is relatively easy to switch the pointer rapidly and repeatedly between different sets of values to simulate more than eight units.

The 6581 chip is referred to as the SID, or Sound Interface Device, and contains functions a great deal more advanced than some of the early purpose-designed music synthesisers. As well as full ADSR control over the volume envelope of each sound, the functions include filtering, different waveforms and ring modulation — modifying one sound with another.

A

ACCESS TIME

This refers to the time taken to locate a particular item from within a whole collection of data. The term is most commonly used when referring to the length of time needed to locate any particular record within a file — especially in database applications. For many business applications, the efficiency of a program will be far more strongly determined by the average access time of the disk than by the clock speed of the CPU.

The access time is quite different from the 'data transfer rate' — which is the speed at which bits will be transmitted from disk to computer once the item has been found. On the Sinclair Microdrive, for example, the average access time of a piece of data is 3.5 seconds. The minimum is zero, if the data is opposite the read-head mechanism when the request is made; and the maximum is seven seconds, if it has to wait for a complete circulation of the tape loop. This is very slow when compared with a floppy or hard disk unit, where the average might be nearer to half a second. However, the data transfer rate of the Microdrive (16 Kbytes per second) is very fast, and is as good as any disk.

ACCUMULATOR

Inside a microprocessor or CPU there are several registers. These are individual bytes of memory that perform all the arithmetic and logical functions of the processor. Probably the most active and important of these is the accumulator, which is linked directly into the Arithmetic Logic Unit (ALU). The chief function of the accumulator is its ability to accumulate values: that is to say the contents of a byte can be simply added into, or subtracted from, this register. To the BASIC programmer, the accumulator is both invisible and inaccessible (although it will be used by the BASIC interpreter thousands of times every second). To the machine code programmer, however, the majority of instructions in every program written will involve some manipulation of the accumulator.

ACOUSTIC COUPLER

The transmission characteristics of a telephone line are such that it can only be used to transmit frequencies in the range 300 Hz to 3400 Hz — the range required to transmit normal speech intelligibly. This 'bandwidth' also determines the maximum rate at which binary data can be transmitted. Some system is needed, therefore, to ensure that the signal to be sent always falls within this range. This is called 'modulation'.

One system of modulation represents a binary zero as a tone in one frequency (let's say 1000 Hz), and a binary one is represented by another tone in a different frequency (e.g. 2000 Hz). The device for converting between binary data signals and these audio frequencies is called a 'modem' (MOdulator/DEModulator). For best results the modem should be wired directly into the telephone line, but this can only be done for a permanent installation. For portable applications (such as salesmen transmitting the day's figures back to central office, or journalists sending copy to their editors) an acoustic coupler is necessary.

An acoustic coupler is simply a modem with two rubber cups (one for the mouthpiece and one for the earpiece) into which a telephone handset is pushed. Were you to remove the handset during a transmission, you would be able to hear the data being transmitted in the form of tones. However, by interrupting the flow of data, you would create errors in the received data.

ACRONYM

BASIC is an acronym, so is PET, and FIFO, RAM, EPROM and SNAFU. An acronym is a word formed by taking the initial letter from each word in a description or title. Acronyms seem to be very popular in the computer industry, both for buzzwords and for proprietary names for products. One suspects, however, that often the final acronym has been thought up first, and then the component words have been fitted to that. Who would really want to call a programming language Beginner's All-purpose Symbolic Instruction Code, or a new computer the Locally Integrated Software Architecture?

ADA

In the late 1970s, C.I.I. Honeywell Bull in France designed and specified a programming language primarily for use by the U.S. Defense Department. It was intended to replace all the other programming languages they were using at the time, and was therefore also intended to vary as little as possible between machines. The language is very highly structured — it is described by some as a kind of super PASCAL, but by others as 'unwieldy'. It is named in honour of Countess Ada Lovelace, who was a close friend and companion of Charles Babbage and is credited with being the first programmer.

THE SPECTRUM OF ZX BASIC

BASIC has become the standard language of microcomputers, but almost every machine has its own variation — or dialect. In this series of articles we will be looking at some of these variations and their functions, as well as explaining how they can be 'translated' from one dialect to another. This first article looks at the most widely used dialect — Sinclair BASIC.

We begin with variable names — always a source of confusion between BASIC dialects. In Sinclair BASIC, string variable names must have only one letter, and there is no distinction between upper and lower case letters. This means that the variables a$ and A$ refer to the same memory location. String array names follow the same rules as simple variables, and pre-empt them, so that once you've DIMensioned the string array H$, all further mentions of H$ in the program will be taken as referring to the array H$. This follows from the fact that Sinclair BASIC regards all string variables as array-type variables, some of them formally DIMensioned, and others not.

Numeric variable names are less constrained than those of string variables: they must begin with a letter, and they must consist of letters or digits, but they may be any length. They may include spaces, and they may be a mixture of upper and lower case letters, but although these factors are helpful to the programmer, they are of no significance to the machine, which will ignore them. Some valid numeric variable names are:

```
qwert, ub40, advanced computer course
```

and the following are exactly equivalent:

```
QWERT, UB 40, Advanced Computer Course
```

Numeric array names must be single letters, but this does not preclude numeric variables of the same name: the array variable v(8) is quite distinct from the simple numerical variable v. Single-letter non-array numerical variables such as v must be used as the counters of FOR...NEXT loops, so FOR V=1 to 9...NEXT V is legal, but FOR loop=1 TO 9 is illegal.

The main differences between the Sinclair dialect and other BASICS lie in the treatment of string quantities. Let us start with the effect of the DIM statement. In Sinclair BASIC, when the statement DIM a$(12) is executed, 12 bytes of memory are set aside exclusively for the use of the variable a$, and these bytes are initialised with spaces. Each of these bytes can be referred to as a subscripted variable, or the whole 12 bytes can be referred to collectively as a$. The length of this variable will always be 12, and assignments to it will be padded with spaces or truncated on the right as necessary to preserve this length. Suppose we write:

```
DIM a$(12): LET a$="123456789"
```

then a$ will actually contain the characters '123456789' followed by three spaces, making 12 characters in all. If we write instead:

```
DIM a$(12):LET a$="ABCDEFGHIJKLMN"
```

then a$ will actually contain only the 12 characters 'ABCDEFGHIJKL' — the string quantity 'ABCD EFGHIJKLMN' has been truncated on the right to fit into the DIMensioned length of a$. If we now write:

```
LET a$(2 TO 5)="1234"
```

then a$ will contain 'A1234FGHIJKL'. This shows the power of Sinclair string handling — all strings are treated as single-dimension string arrays, the arrays can be subscripted or not, and individual elements of an array can be accessed — singly or as part of a sub-string — by subscripts. It also shows another major divergence from other versions of BASIC. Elsewhere DIM a$(12) creates 12 separate string variables called a$(1), a$(2), etc., each of which has the length of the expression assigned to it. If nothing has been assigned to a particular string variable, then its length is 0, and it contains only the null string, "".

In other BASICS this way of handling strings requires the various string functions, LEFT$, RIGHT$, MID$, and sometimes INSTR, to enable sub-string manipulation and string-slicing in the way demonstrated. But this is not so in Sinclair BASIC. The Sinclair equivalents of these string functions are:

```
LEFT$(A$,N) = A$( TO N)
```

(meaning the N leftmost characters of A$);

```
RIGHT$(A$,N) = A$(LEN A$-N+1 TO )
```

(meaning the N rightmost characters of A$); and

```
MID$(A$,P,N) = A$(P TO P+N-1)
```

(meaning the N characters from position P onwards in A$).

```
LET S=INSTR(A$,"teststring")
```

(meaning find the starting position in A$ of the substring "teststring") can be replaced by:

```
LET Y$=A$:LET Z$="teststring":GOSUB 9900:LET
   S=POSN
9900 LET ZL=LEN Z$:LET SL=LEN Y$-ZL+1:LET
```

```
    POSN=0
9910 FOR K=1 TO SL
9920 IF Y$(K TO K+ZL-1)=Z$ THEN LET POSN=K:LET
    K=SL
9930 NEXT K:RETURN
```

Notice in this subroutine that the string variable Y$ is treated as a subscripted array-type variable, even though it has not been DIMensioned. Since in Sinclair BASIC all string variables are array-type variables, a string variable that is not DIMensioned is implicitly a variable-length single-dimension array of single characters; if it is DIMensioned, its element length is fixed by the last number in the DIM statement. Whereas in other BASICS DIM x$(8,7) creates a two-dimension array, in Sinclair BASIC it creates a single-dimension array of eight elements, each of them fixed in length at seven characters.

The strict attention paid to the length of DIMensioned string variables by Sinclair BASIC means that seemingly simple statements can have differing effects, depending upon whether a DIM statement has been executed or not. If a$ is a simple string variable, then LET a$="" makes the contents of a$ equal to the null string ("") and the length of a$ equal to zero. If DIM a$(7) has been executed previously, however, then LET a$="" makes the contents of a$ equal to seven spaces, and the length of a$ equal to seven (which it will always be, following the DIM statement). Furthermore, in such a case, even though LET a$="" has been executed, a test such as:

```
IF a$="" THEN PRINT "null-string"
```

will fail, and nothing will be printed — a$ is equal to seven spaces, not the null-string.

If you need to test string array elements in this way, then it's probably best to set aside a string variable for the purpose, DIMension it to the length of the longest array variable used in the program, and test your array variables against it, like this:

```
100 DIM a$(12,34)
120 DIM b$(7,56)
140 DIM N$(56)
150 REM N$ will be used as the empty string
. . . . . .
580 IF b$(3)=N$( TO 56) THEN PRINT "empty"
590 IF a$(11)=N$( TO 34) THEN PRINT "empty"
. . . . . .
```

Here N$ is used only as the empty string, and if it wasn't used in this way then the tests in lines 580-590 would have to use literals, thus:

```
580 IF b$(3)="                    " THEN PRINT "empty"
585 REM 56 spaces between the quotes
```

This is inconvenient and prone to error. An alternative to using N$ in this way is to DIMension all array variables with one more element than they need, and use that last element as an empty string for tests of that array, so that line 590 might be:

```
590 IF a$(11)=a$(12) THEN PRINT "empty"
```

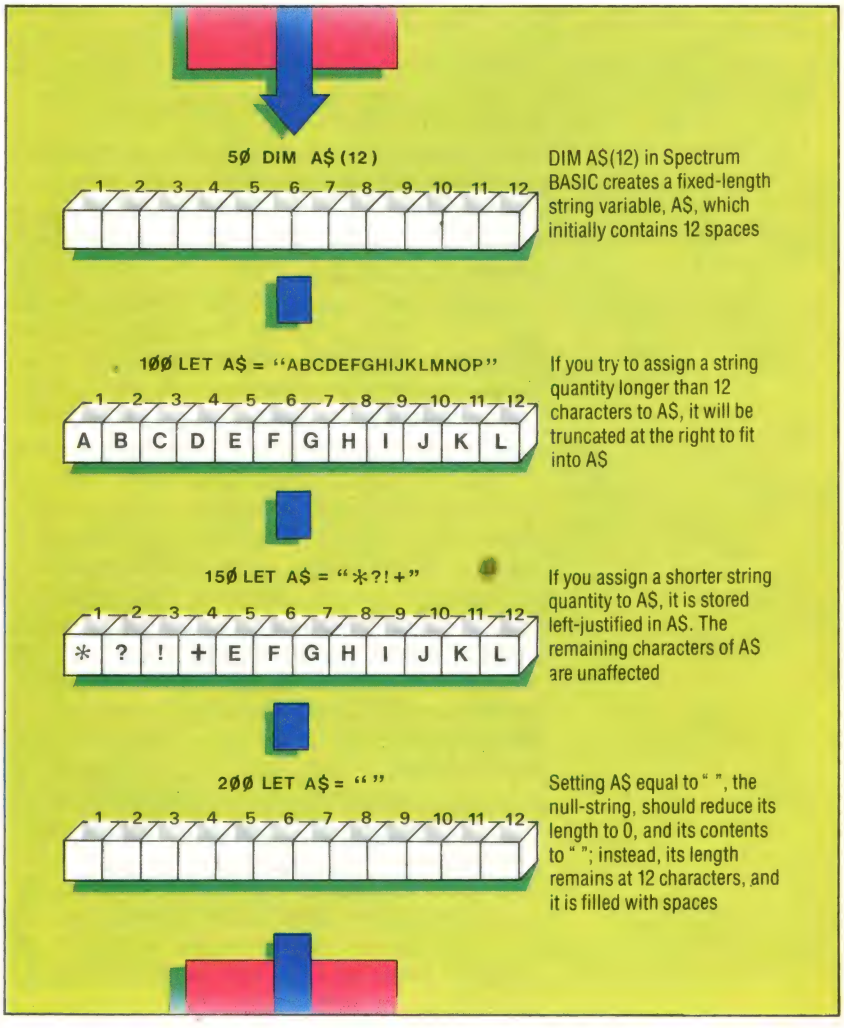
IAN McKINNELL

SuperBASIC
While Sinclair's SuperBASIC has a considerably enhanced range of commands over ZX BASIC, the most significant feature is its abandonment of the single-key reserved word entry system common to the ZX80, ZX81 and Spectrum. This was originally introduced as an economy measure for users (it was felt that pressing a single key rather than typing a whole word would prove attractive). The system dictated that a variety of different 'modes' would be necessary to allow the entry of single characters to be differentiated from the entry of key words. This system was attractive to Sinclair users who had never previously encountered a keyboard, but for those who had used a typewriter it proved to be a source of frustration

assuming that a$(12) is never used and so contains only spaces.

Notice lastly that, in Sinclair BASIC, the first element in any array has the subscript one, whereas in some other BASICS the first element in an array has subscript zero. In the next instalment of the course, we will conclude this look at Spectrum BASIC.

Procrustean Strings
The mythical Greek character Procrustes was an innkeeper who kept only one size of bed, and stretched or truncated his guests to fit it


5Ø DIM A$(12)
1 2 3 4 5 6 7 8 9 10 11 12
DIM A$(12) in Spectrum BASIC creates a fixed-length string variable, A$, which initially contains 12 spaces
1ØØ LET A$ = "ABCDEFGHIJKLMNOP"
1 2 3 4 5 6 7 8 9 10 11 12
A B C D E F G H I J K L
If you try to assign a string quantity longer than 12 characters to A$, it will be truncated at the right to fit into A$
15Ø LET A$ = "*?!+"
1 2 3 4 5 6 7 8 9 10 11 12
* ? ! + E F G H I J K L
If you assign a shorter string quantity to A$, it is stored left-justified in A$. The remaining characters of A$ are unaffected
2ØØ LET A$ = ""
1 2 3 4 5 6 7 8 9 10 11 12
Setting A$ equal to "", the null-string, should reduce its length to 0, and its contents to ""; instead, its length remains at 12 characters, and it is filled with spaces

KEVIN JONES

INTRODUCING FIRST CONCEPTS

Machine code programming is the key to the real power of the microprocessor, allowing the programmer direct control over all the machine's functions. This first part of a comprehensive course, covering both 6502 and Z80 operation codes, will lead to a full understanding of the fundamentals of computer programming.

Machine code is a programming language, and it looks like this:

```
INSTK: SBC $D9FA,X ;Outport flag value
```

or like this:

```
DE23 FD FA D9
```

or like this:

```
11011110 00100011 11111101 11111010 11011001
```

Sometimes it looks like this:

```
1240 LET ACC=ACC-FLAG (X)
```

and sometimes like this:

```
PERFORM FLAG-ADJUST THROUGH LOOP1
```

It's all code of a sort, and since it's destined for a computing machine it's called *machine code.* To the machine it doesn't actually *look* like anything at all, being simply a pattern of voltage levels or a current of electricity.

What we usually mean when we say machine code is Assembly language, and the first example we gave in this article is an instruction in 6502 Assembly language. The point of giving all the other examples was to demonstrate that there is no specific machine language as such, only a number of different ways of representing a sequence of electrical events, and representing them in ways that we find more or less easy to understand. So the first thing to learn about machine code (or Assembly language – we won't worry about the distinction for the moment), is that it's just another programming language. However, the programming must always come before the language: whether you write your programs in IBM Assembler, Atari BASIC, or Venusian PsychoBabble, you have to solve the programming problem in your mind before you touch a keyboard. The programming language in which you then express your solution will obviously influence the form of the final program. Indeed you may choose among various possible languages precisely to make the coding of your program easier, or shorter, or more readable. But the solution must always come first: content must precede form.

In that case, why call it machine code, and why bother to use it at all? We give the language this name because its instruction set corresponds exactly with the set of 'primitive' or fundamental operations that a particular microprocessor can perform. We use the machine code when it is important to direct the operation of the microprocessor exactly, step-by-step, rather than allowing a program language interpreter to control it in a more general way.

The commonest reason for wanting to use it is speed: if your program addresses the processor more or less directly, then you avoid the relatively lengthy business of program translation. In other words, by cutting out the middleman you save time. Program execution time, that is. The actual coding, testing, debugging, modification and maintenance of a machine code program is likely to take at least twice as long as the same operations would on a high-level language program. The unfriendliness and intractability of machine code stimulated the development of languages such as COBOL and BASIC.

If the set of machine code instructions is the set of processor operations, then what are these operations, and what does the processor do? In the simplest terms the Central Processing Unit (CPU) of a computer is a switch that controls the flow of current in a computer system between and among the components of that system. Those components are the memory, the Arithmetic Logic Unit (ALU), and the Input/Output devices. When you press a key on the keyboard, you are inputting some information; in the machine, however, you are simply generating a pattern of voltages in the keyboard unit. The CPU switches that pattern from the keyboard to part of the memory, then switches a corresponding pattern from elsewhere in memory to the screen so that a character pattern appears on the screen. To you this process may seem like operating a typewriter, but in a typewriter there is a mechanical connection between hitting a key and printing a character, whereas in a computer that linkage exists only because the CPU switches the right voltage patterns from place to place. Sometimes pressing a key doesn't cause a single character to appear on the screen: the keypress may destroy an asteroid, or save a program, or delete a disk file, or print a letter. The operation depends on how and where the CPU switches the electric current.

In this simplistic view the CPU is at the heart of the system, and all information (or electrical current) must pass through it from one component

to another. In fact, the CPU and the system are more complicated than that, but it's not a misleading view. You can think of the CPU as a master controller that sets lesser switches throughout the system to control the flow of electricity, and thus controls the flow of information indirectly, rather than routing all information physically through itself.

The effects of the CPU's switching operations can be classified for our purposes as: arithmetic operations, logical operations, memory operations, and control operations. These operations are all the results of switching information through different paths in the system and in the CPU, and to the CPU they all seem like the same sort of thing.

Arithmetic operations are really the most important feature of the machine. The CPU can add two numbers together, or subtract one from the other. Subtraction is achieved by representing one of the numbers as a negative number and adding that negative number to the other number; 7+5=12 really means:

(plus 7) added to (plus 5) equals (plus 12).

7−5=2 really means:

(plus 7) added to (minus 5) equals (plus 2).

Multiplication and division are regarded as repeated additions or subtractions, so the CPU can be programmed to simulate these processes as well. If the CPU can cope with the four rules of arithmetic, then it can cope with any mathematical process. It is well to remember, however, that all its mathematical potential relies on the ability simply to add two numbers.

Logical operations for our present purposes can be described as the ability to compare two numbers: not merely in terms of relative size, but also in terms of the pattern of their digits. It's easy to see that seven is bigger than five because we can take five away from seven and still have a positive result. The CPU has the ability to do that sort of comparison, and it can also compare 189 with 102 and recognise that both numbers have the same digit in the hundreds column. It may not seem a very useful ability as yet, but its use will become more evident later.

The CPU can perform essentially two memory operations: it can copy information from a memory location into its own internal memory, and it can copy information from its internal memory to another memory location. By doing these two things one after another it can therefore copy information from any part of memory to any other part of memory. For the memory to be any use, the CPU must be able to do both these things, and these two operations are all it needs for complete management of the memory.

Control operations are really decisions about the sequence in which the CPU performs the other operations we have described here. It's not important at the moment to understand them any better than that: if you accept that the CPU can make decisions about its own operation, then that is sufficient at this stage.

So the CPU can do arithmetic, it can compare numbers, it can move information around in memory, and it can decide its own sequence of operations. This is a simple list of procedures, and yet it completely descibes or specifies an ideal computing machine! If the CPU can do those four things, then by doing them in the right sequences it can perform any computable task. The right sequence, of course, is the computer program for the particular task, and that's where we as programmers come in. If the CPU had the ability to generate its own operation sequences, then there would be no need for us.

You may not be convinced that the four types of operation we have described are a sufficient description of a conceptual computer, so let's think about a BASIC program in terms of the general operations performed. What are these fundamental operations? In any program you have variables, which are simply the names of places in memory where information is stored. Most programs perform some sort of arithmetic upon some of these variables. Having done the arithmetic, a program will often compare two pieces of information and as a result will execute one set of instructions or another. Information usually comes into a program from the user at the keyboard, and goes out to the user via the screen.

Except for the sentence about input and output, this description contains no more than the four elemental CPU operations put into different words. And, if you accept for the moment that to the CPU all Input/Output devices are just special areas of memory, then the picture of the ideal computer executing actual programs is complete. Consequently, the execution of a program can be described as a directed flow of information into, around, and out of the computer; you supply some information via the keyboard, that information is manipulated by your program, and some information appears on the screen.

If the idealised computer is just a CPU and some memory, then before going any further we should investigate computer memory: what is it, and how does it work?

Imagine a simple electrical circuit consisting of a battery, a switch, and a light bulb: if the switch is closed the light goes on, and stays on until the battery runs down or until the switch is opened. Then the condition of the light bulb — ON or OFF — is a piece of information, and the whole circuit is a memory device recording that information. Suppose now that the switch is placed at the entrance to a factory, and the light is placed in the Manager's office. When the first employee arrives at the factory, he or she closes the switch at the entrance, and the Manager in the office can see that the light is on and therefore knows that someone has turned up for work. The Manager doesn't have to be in the office when the light goes on; he or she can look at the light bulb at any time to find out whether someone has

arrived. The information that someone has turned up for work is stored in the circuit.

That's almost exactly how information is stored in computer memory: all information reduces to the presence or absence of electricity in a circuit. Naturally there's more to it than that, so let's improve the management information system. Suppose we have four separate switch/bulb circuits (the four switches in a row at the door, and the four bulbs in a corresponding row in the office), so that closing the leftmost switch illuminates the leftmost bulb, and so on. Now imagine that every employee is told to close the switches in a unique way, so that when Catherine arrives she closes the first and second switches and opens the third and fourth; Richard closes the fourth switch and opens all the others; Bobby closes the first and third and opens the second and fourth; and so on for all the employees. The lights in the office now show the Manager which of the employees has turned up for work.

Suppose that the OFF position of each switch is labelled 0, and the ON position is labelled 1: therefore Catherine has to set the switches 1100 (first two switches ON, third and fourth OFF), Richard has to make the pattern 0001 (fourth switch ON, the others OFF) and Bobby has to set 1010 (first and third ON, the other two OFF). If the Manager reads each light bulb as 1 if it's ON, and 0 if it's OFF, then both the employees and the Manager will be speaking the same identification language. '0001' means 'Richard' to both people.

How many unique patterns of switches are there? Each switch can be in one of two positions, and there are four switches, so there are $2\times2\times2\times2=16$ different patterns. Let's consider all the possibilities:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

Try as you like, you can't make any more patterns than these, and there are 16 of them.

Notice how quickly we've moved from the concrete picture of light bulbs in a room, to the abstract matter of patterns of 1's and 0's. If we can abstract a little further we can turn these patterns into numbers.

Think about counting and writing down as you count. You can count from nought to nine very easily because each of those numbers has a unique name and a symbol to represent it. But what do you write down after nine? You have a name, ten, for that number, but no separate symbol to represent it. Therefore you must re-use some of the other symbols: 10, 11, 12, and so on until 99, when you run out of possibilities again, so the next number has three columns (100). This seems trivial, but you may remember how difficult it was when you learned it at school: all that squared paper with Hundreds Tens and Units written at the top of each sum? You now know that the number 152 means "1 in the Hundreds, 5 in the Tens, 2 in the Units", or $100+50+2=152$. Counting works like this because we have ten digits (0,1,2,3,...,9) which we arrange to represent all possible numbers.

How does counting work, however, if there are only two digits: 0 and 1? We can count to 1 easily, but how can we represent the next number? We have run out of unique digits, so we must re-use what we have (just as we did when counting with ten digits), and write the next number as 10. Now we know that the next number is called 'two', so in this system 10 represents the number two. The next number as we count is three, and we must write that as 11. Then what? We've run out of two-digit combinations, so the next number, four, must be represented as 100; five must be 101, six is 110, and seven is 111. Here, we are counting in decimal numbers (nought, one, two, etc), but we're writing these down in binary numbers (0,1,10,11,100,101,...).

In the same way as a decimal number such as 152 means: $(1\times100)+(5\times10)+(2\times1)$, the binary number 101 means: $(1\times4)+(0\times2)+(1\times1)$. Instead of having hundreds, tens, and units columns for our numbers, we must use columns marked: fours, twos, and units. In a decimal number the value of a digit is multiplied by ten for every column it moves to the left; in a binary number the value of a digit is multiplied by two for every column it moves to the left.

So that's the binary system: just a different way of representing numbers. If you know Roman numerals you don't find it hard to accept that there are VII dwarfs in *Snow White*; so why not write 111 dwarfs? The actual number of dwarfs is not changed by the way we represent it, but it is a good idea to say the binary number as 'binary one one one', and to write it as '111 b' so that you don't confuse it with a decimal representation.

Now we can return to our original analogy of how the factory workers switch patterns, and decide on a method of making these a little easier to use. The most sensible thing to do is to treat these patterns as four-digit binary numbers. This means that Catherine's signal is 1100 binary, which is 12 decimal. Richard's signal is 0001 binary (1 decimal), and Bobby's signal is 1010 binary (10 decimal). When the Manager looks at a pattern of lights in the office, he or she can read it as a binary number, convert it to its decimal equivalent, and look down the list of employees to see who that number corresponds to. Thus we can say that information is stored in the current of electricity, and the switches make it meaningful.

Our analogy has given a simple picture of how information is represented in a computer: to the computer it's just patterns of voltages (i.e. lights are ON or OFF), but we humans find it easier to consider those patterns as binary numbers. It's all a matter of representation. If you now think of 1010 as the code meaning 'Bobby', then you may start to see how all of this relates to machine code itself. In the next instalment of the Machine Code course, we will look at how binary numbers are used to represent information inside your home computer.

Speeding Ahead
These three short programs, one for ZX Spectrum, one for the BBC Micro and the other for the Commodore 64, demonstrate the difference in speed of operation between BASIC and Machine Code by displaying either the entire character set (Commodore and BBC), or colour blocks (Spectrum), on the screen

Spectrum

```
1 REM*************************
10 REM***SPECTRUM M/C CODE****
11 REM* DO NOT LIST LINE 1   *
12 REM* AFTER RUNNING PROG   *
13 REM*************************
99 REM*************************
150 LET PTR=23635:LET SA=PEEK
(PTR)+(256*PEEK(PTR+1))+7
200 BORDER 2
350 DATA 1,0,3,17,0,88,33,0,0,
237,176,201
400 FOR X=0 TO 11
500 READ MC
600 POKE SA+X,MC
700 NEXT X
1000 LET OFFSET=0
1100 FOR X=0 TO 1 STEP 0
1200 POKE SA+7,OFFSET
1300 LET DUMMY=USR SA
1400 LET OFFSET=OFFSET+13
1500 IF OFFSET>=256 THEN LET
OFFSET=OFFSET-256*INT(OFFSET/256)
1600 NEXT X
1700 STOP
1799 REM**********************
1800 REM*SAVE PROG BEFORE RUN*
1801 REM*SAVE PROG BEFORE RUN*
1802 REM**********************
```

BBC Micro

```
100 REM*********************************
*******************BBC***************
149 REM*****************************
150 REM*      BBC M/C CODE DEMO    *
151 REM*****************************
200 MODE 4:*TV 254
300 GOSUB 30000
700 FOR P=1920 TO 6079
800 K=K+1:IF K>2679 THEN K=1920
900 ?(HIMEM+P)= (K+47232)
1000 NEXT P
1100 PRINT TAB(13);"THAT WAS BASIC"
1200 INPUT" HIT RETURN FOR MACHINE CODE
VERSION ",A$:CLS
1300 FOR L=0 TO 15:FOR B=0 TO 255 STEP L
1400 ?(SA)=LS:?(SA+1)=HS
1500 ?(FA)=LF:?(FA+1)=HF
1600 DUMMY=USR(PSTRT)
1700 VDU 30
1800 NEXT B,LP
1900 STOP
30000 REM******MC LOADER S/R****
30010 K=1919:PSTRT=PAGE+8:VSTR=HIMEM+192
0
30020 HS=INT(VSTR/256):LS=VSTR-256*HS:LF
=LS+56:HF=HS+2:SA=114:FA=116
30100 DATA 50,169,32,197,112,48,4,240,2,
133,112,165,112,32,227,255
30110 DATA 230,114,208,2,230,115,165,116
,197,114,208,7,165,117
30120 DATA 197,115,208,1,96,230,112,169,
128,197,112,208,224
30130 DATA 169,32,133,112,208,218,96,96,
96
30150 READ ZZ
30160 FOR BY=PSTRT TO PSTRT+ZZ
30170 READ MC:?(BY)=MC
30180 NEXT BY
30200 RETURN
30299 REM*****************************
30300 REM*   SAVE BEFORE RUNNING !! *
30301 REM*****************************
30399 REM*****************************
30400 REM*DO NOT LIST LINE 100 AFTER*
30401 REM*      RUNNING PROGRAM !!   *
30402 REM*****************************
```

Commodore 64

```
99 REM ***************************
100 REM*COMMODORE M/C CODE DEMO *
101 REM***************************
200 PRINT CHR$(147)   :REM CLEAR SCREEN
300 PRINT "     THIS WON'T TAKE LONG"
400 GOSUB 60000
500 PRINT CHR$(147);CHR$(5) :REM CLS AND
 WHITE
600 CC=0
700 FOR P=SM TO FM
800 POKE P,CC:POKE P+OF,CL
900 CC=CC+1:IF CC>255 THEN CC=0
1000 NEXT P
1100 PRINT TAB(13);"THAT WAS BASIC"
1200 INPUT" HIT RETURN FOR MACHINE CODE
VERSION ";A$
1300 FOR LP=1 TO 9:FOR B=0 TO 255 STEP L
P
1400 POKE SA,LS:POKE SA+1,HS
1500 POKE FA,LF:POKE FA+1,HF
1600 POKE BA,B:POKE CH,0
1700 SYS AA
1800 NEXT B,LP
1900 STOP
60000 REM******MC LOADER S/R****
60010 SM=256*PEEK(648):OF=55296-SM:FM=SM
+999:BD=53280:SC=BD+1:CS=8:CB=6:CL=0
60020 POKE BD,CB:POKE SC,CS
60030 LS=0:HS=PEEK(648):LF=232:HF=HS+3:S
A=251:FA=253:BA=250:CH=2
60100 DATA 850,885,169,0,170,165,250,133
,2,165,2,129,251
60110 DATA 230,251,208,2,230,252,165,253
,197,251,208,7,165,254
60120 DATA 197,252,208,1,96,230,2,208,22
9,240,223
60150 READ AA,ZZ
60160 FOR BY=AA TO ZZ
60170 READ MC:POKE BY,MC
60180 NEXT BY
60200 RETURN
60299 REM*******************************
60300 REM* SAVE THIS BEFORE RUNNING IT*
60301 REM*******************************
```

BILL GATES–SETTING THE STANDARD

Microsoft has become, in one short decade, the world's most influential supplier of microcomputer software. It was courted by the world's biggest supplier of computers, IBM, and effectively helped shape the specification of the IBM PC, the world's largest-selling personal computer.

COURTESY OF MICROSCOPE

TONY SLEEP

The Microsoft company, now a multi-million dollar operation, is a classic story of enthusiasts made good. Bill Gates, at 28 the chairman of the board, was in 1972 only a talented amateur.

At Seattle High School, where the parent-teacher association had the foresight to equip the students with a timesharing terminal attached to the popular DEC PDP-11 minicomputer, Bill learned about the workings of computers. He went to Harvard University and on his graduation went into business back in Bellevue with schoolfriend Paul Allen. The firm they set up was called Traff-O-Data, and their work was to monitor traffic flow for the Seattle public authorities. It was a momentous period in the development of the microcomputer: the first microprocessors were making an appearance and those with imagination and enthusiasm saw a great future for devices such as Intel's 4004 and, later, 8008 chip. Bill was by now thoroughly familiar with the DEC PDP-11 and one of his first jobs was to track down bugs in this computer. It occurred to him that it would be a good idea to adapt its BASIC for use on the 8080. He had no development system, and the first occasion on which the code and the machine were mated was when Gates took the tapes down to Altair in Albuquerque, New Mexico. Incredibly, it ran first time. Thus was born MBASIC, which has ever since been the standard to beat.

Microsoft was becoming known as a software house with expertise at fitting new computers with operating systems – filling the empty box, as it were – and IBM contacted Gates to ask for his advice on how to specify and equip a single-user personal computer. Initially, Gates suggested that Gary Kildall of Digital Research, riding high on the burgeoning success of CP/M, was the man for the job. But eventually IBM came back to Microsoft. Microsoft rewrote PASCAL, FORTRAN and MBASIC for the 16-bit implementation, and also came up with the GW (for 'gee-whizz') BASIC with its extended music and graphics capabilities.

At the same time, Gates realised that an untidy but powerful multi-user OS by Bell Laboratories could be usefully adapted for the more powerful micros based on the new 16/32-bit microprocessors, and transformed Unix into Xenix. Both Tandy and Apple adopted Xenix in their own 16/32 bit models in 1983. It even transpires that Microsoft did much of the work for Apple's newest creation, Mackintosh.

Microsoft has a firm footing in the hobby market, too. In 1981 it set up ASCII-Microsoft with a keen young Japanese, Kay Nishu, to sell their OS and BASIC to far Eastern manufacturers of the new generation of lap-held micros like the NEC PC 8201 and Tandy Model 100. Out of the Japanese manufacturers' desire for a common standard, not only in languages, but in interfaces to desirable home peripherals such as colour plotters and printers, lightpens, joysticks, trackballs, robot arms, FM tuners and so forth, came the common MSX standard. Now, it seems, we shall soon have a standard common disk format from Microsoft that will enable data to be transferred among the three principal operating environments – MSX, MS-DOS, and Xenix. With its emphasis on software that is easy to use, illustrated in such phenomenal advances as screen windows and the mouse, Microsoft would appear to have a bright future ahead of it.

GUIDING PRINCIPLES

In 1970, at the age of 28, Shiina Takayoshi abandoned a promising career in the military and formed the Sord Corporation (1982 sales: $40 million). He immediately formulated 11 guiding principles to help him govern his new computing business. These included:

***The company's foremost obligation is to humanity.**

***The company must do its best to determine what products and services are best for society, and provide these at a reasonable cost.**

***There must be no division between labour and management. All persons in the company must respect one another and co-operate for the benefit of all**

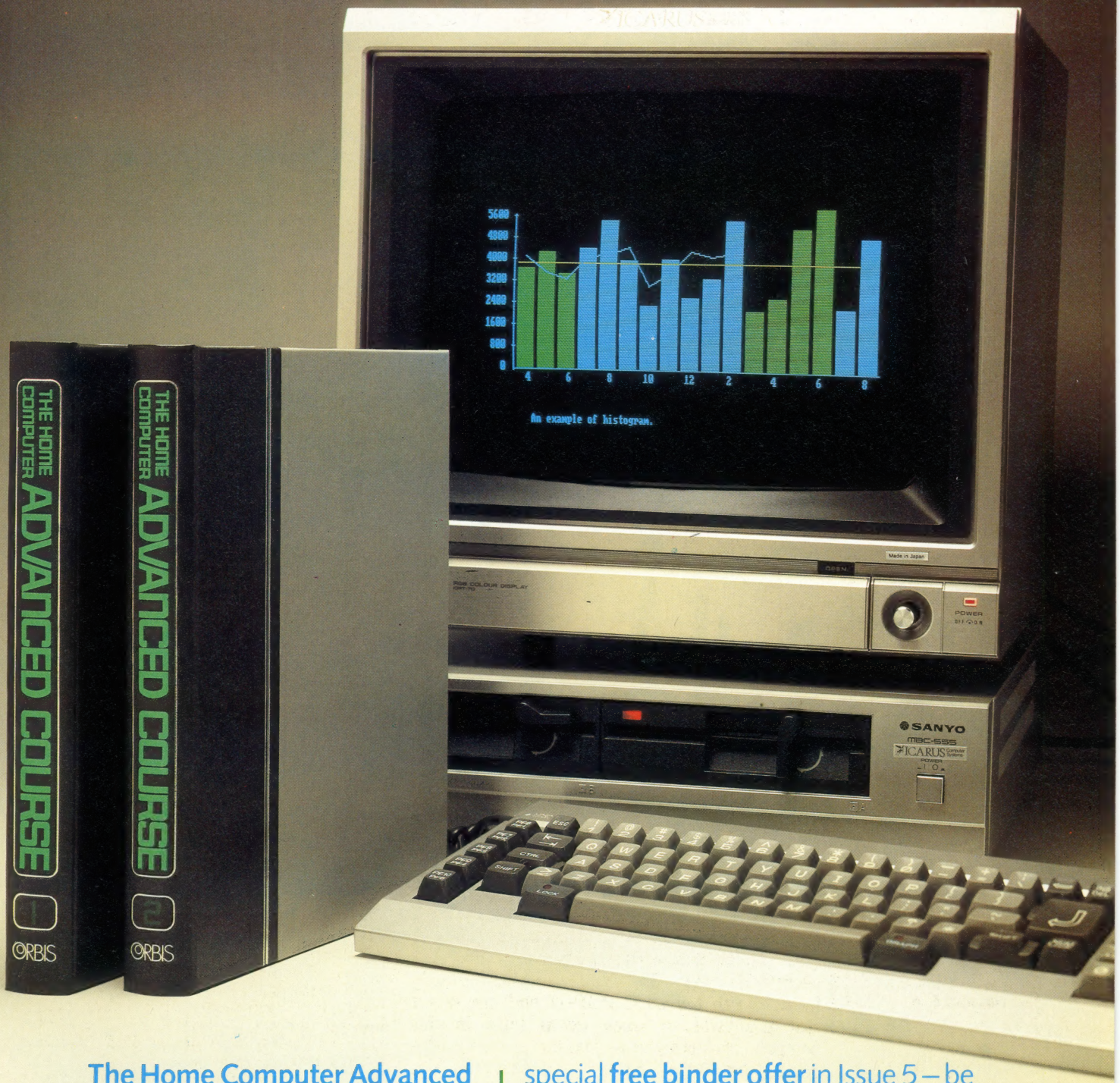
Industry Standard

BASIC — Beginners' All-purpose Symbolic Instruction Code — was developed in 1965 at Dartmouth College, US, by J Kemeny and T Kurtz, and thus predates the microprocessor by at least seven years. While many dialects of this language have been formulated, MBASIC, Microsoft's own version, has come to be recognised as the industry standard.

Microsoft established its reputation with the success of MBASIC, and has continued to thrive by producing a serious challenger to Digital Research's CP/M in MS-DOS, an operating system designed to be applicable to a wide range of microcomputers.

Following the lead given by Xerox's Star terminal system, and developed by Apple with Lisa, Microsoft has now diversified slightly and produced a package that combines software with a hardware device necessary to its operation — MS-WINDOWS and the mouse. Microsoft's mouse, like that of its two competitors, uses a trackball-like arrangement coupled with two selectors to move the cursor around the screen

The volume 1 binder can be yours free!



The Home Computer Advanced Course will take you far beyond the novice stage, widening your knowledge and making you a more sophisticated user.

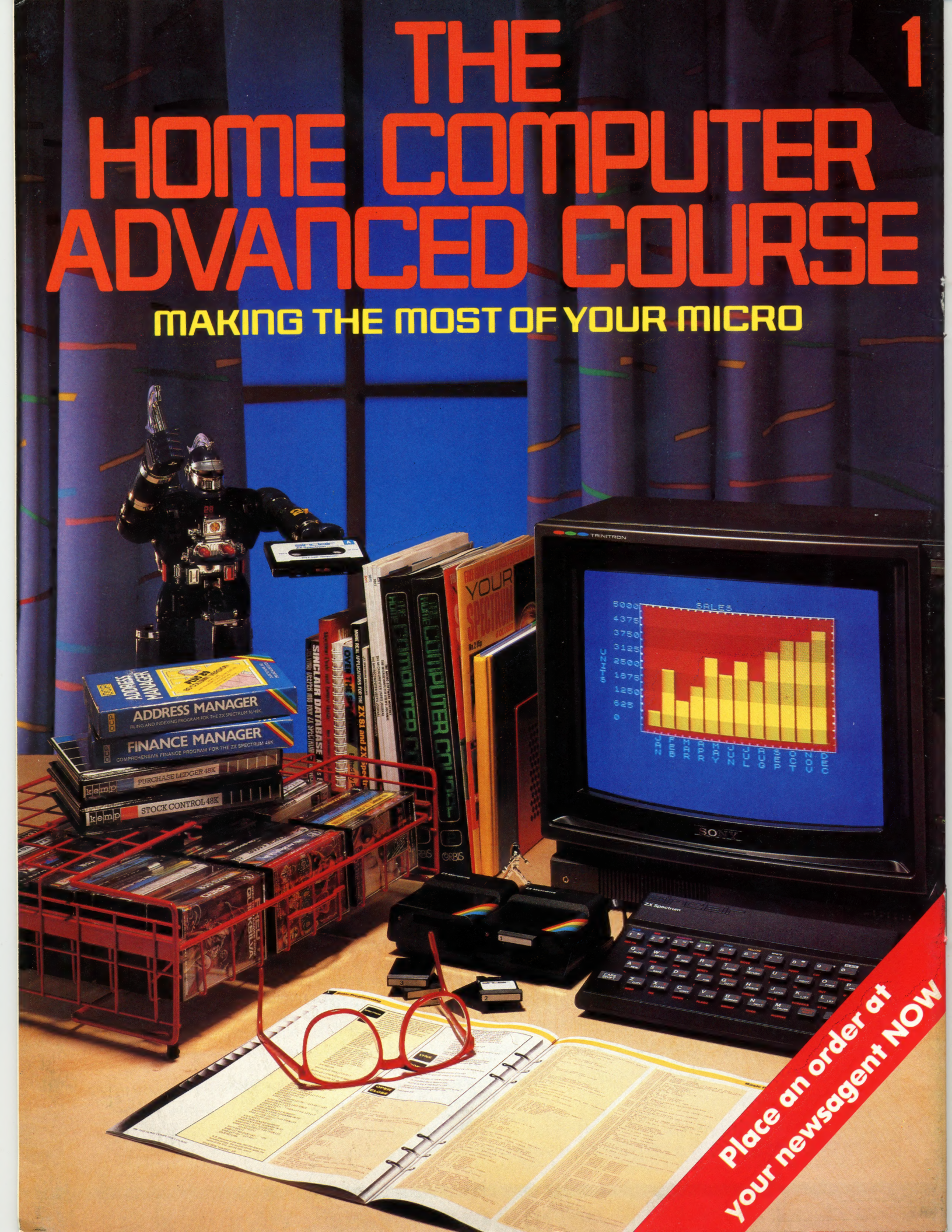
To help you keep your copies immaculate, we will be making a very special **free binder offer** in Issue 5 – be sure not to miss it!

Overseas readers: this special offer applies to readers in the U.K., Eire and Australia only. Binders may be subject to import duty and/or local tax.

THE HOME COMPUTER ADVANCED COURSE

1

MAKING THE MOST OF YOUR MICRO



Place an order at your newsagent NOW